НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет Інформатики та обчислювальної техніки

Кафедра Обчислювальної техніки

До захисту допущено:
Завідувач кафедри
_____ Сергій СТІРЕНКО
«___»_____ 2020 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Комп'ютерні системи та мережі»

спеціальності 6.050102 « Комп'ютерна інженерія »

на тему: «Агрегатор новин, реалізований у вигляді WEB-порталу»

Виконав:
студент IV курсу, групи ІО-63
Надолінський Олексій Геннадійович _____

Керівник:
ст. вик. Антонюк Андрій Іванович _____

Консультант з нормо контроль:
проф., д. т. н. Сімоненко Валерій Павлович _____

Рецензент:
доцент, к.т.н. Пасько Віктор Петрович _____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2020 року

ВІДОМІСТЬ ДИПЛОМНОГО ПРОЄКТУ

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	ІАЛЦ.467200.002 ОА	Опис альбому	1	
3	А4	ІАЛЦ.467200.003 ТЗ	Технічне завдання	3	
4	А4	ІАЛЦ.467200.004 ПЗ	Пояснювальна записка	62	
5	А4	ІАЛЦ.467200.005 Д1	Додаток 1	10	
6	А4	ІАЛЦ.467200.006 Д2	Додаток 2	1	
7	А4	ІАЛЦ.467200.007 Д3	Додаток 3	1	
8	А4	ІАЛЦ.467200.008 Д4	Додаток 4	1	

				ІАЛЦ.467200.001 ТП		
	ПІБ	Підп.	Дата			
Розробн.	Надолінський			Відомість дипломного проєкту	Лист	Листів
Керівн.	Антонюк				1	1
					КПІ ім. Ігоря Сікорського Каф. ОТ Гр. ІО-63	
Н/контр.	Сімоненко					
Зав.каф.	Стіренко					

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет Інформатики та обчислювальної техніки
Кафедра Обчислювальної техніки

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – **6.050102 « Комп'ютерна інженерія »**

Освітньо-професійна програма «Комп'ютерні системи та мережі»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Сергій СТІРЕНКО
«___»_____2020 р.

ЗАВДАННЯ

на дипломний проєкт студенту

Надолінському Олексію Геннадійовичу

1. Тема проєкту «Агрегатор новин, реалізований у вигляді WEB-порталу», керівник проєкту ст. вик. Антонюк Андрій Іванович, затверджені наказом по університету від «07» травня 2020 р. № 1081-С

2. Термін подання студентом проєкту 26.05.2020 р.

3. Вихідні дані до проєкту *технічна документація. теоретичні дані, інтернет публікації на тему роботи*

4. Зміст пояснювальної записки:
– провести аналіз форм представлення оновлен інформації на сайтах новин та огляд існуючих агрегаторів новин;
– обгрунтувати пропонований варіант рішення та вибір інтрументів для його реалізації;
– розробити програмну реалізацію агрегатора новин;
– провести тестування функціоналу розробки, написати інструкцію для користувача;

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслеників, плакатів, презентацій тощо):

узагальнена схема роботи системи, блок-схема алгоритму модуля парсера, uml-діаграма классів бази данних.

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормо контроль	проф., д. т. н. Сімоненко В. П.		

7. Дата видачі завдання 01.09.2019 р.

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
/.	Затвердження теми роботи	*01.09.2019*	*виконано*
2.	Вивчення та аналіз завдання	*10.10.2019-10.12.2019*	*виконано*
3.	Розробка архітектури та загальної структури cucmeми	*15.12.2019-14.01. 2019*	*виконано*
4.	Розробка структур окремих підсистем	*20.01. 2020-15.02. 2020*	*виконано*
5.	Програмна реалізація системи	*05.03. 2020-10.04. 2020*	*виконано*
6.	Оформлення пояснювальної записки	*11.04. 2020-20.05. 2020*	*виконано*
7.	Захист програмного продукту	*22.05. 2020*	*виконано*
8.	Передзахист	*26.05. 2020*	
9.	Захист		

Студент Олексій НАДОЛІНСЬКИЙ

Керівник Андрій АНТОНЮК

АНОТАЦІЯ

Даний дипломний проект присвячений створенню web-порталу для агрегації (сбору з різних джерел) актуальних новин з можливістю фільтрації за декількома параметрами результату агрегації.

Web-портал являє собою інтерактивний сайт, який дозволяє створювати власні підбірки новин з різних джерел інформації. Існує можливість необов'язкової реєстрації для того, щоб зберігати власні підбірки новин. Сайт підтримує основні формати стрічок оновлень, такі як RSS та Atom. В підбірках існує на вибір декілька варіантів фільтрації статей для більшої зручності користувачів.

У даному дипломному проекті розроблено: архітектуру web-ресурсу, алгоритм авторизації користувача, модуль-парсер XML-сторінок у форматі RSS/Atom а також дизайн клієнтського інтерфесу web-порталу.

ABSTRACT

This diploma project is dedicated to the creation of a web-portal for aggregation (collection from various sources) of current news with the ability to filter by several parameters of the aggregation result.

Web-portal is an interactive site that allows you to create your own selection of news from various sources. Optional registration is possible in order to keep your own selection of news. The site supports major update feed formats, such as RSS and Atom. In the selections there are several options for filtering articles for greater user convenience.

In this diploma project the following things were developed: web-resource architecture, user authorization algorithm, RSS / Atom format XML-pages parser module in, as well as web-portal client interface design.

Опис альбому

№	Формат	Позначення	Найменування	Кіл-ть аркушів	№ екз.	Примітка
			Документація загальна			
			Розроблена наново			
	А4	**ІАЛЦ.467200.003 ТЗ**	Агрегатор новин, реалізований	3		
			У вигляді web-порталу			
			Технічне завдання			
	А4	**ІАЛЦ.467200.004 ПЗ**	Агрегатор новин, реалізований	62		
			У вигляді web-порталу			
			Пояснювальна записка			
	А4	**ІАЛЦ.467200.005 Д1**	Агрегатор новин, реалізований	10		
			У вигляді web-порталу			
			Сирцевий код проєкту			
	А4	**ІАЛЦ.467200.006 Д2**	Агрегатор новин, реалізований	1		
			У вигляді web-порталу			
			Структурна схема системи			
	А4	**ІАЛЦ.467200.007 Д3**	Агрегатор новин, реалізований	1		
			У вигляді web-порталу			
			UML діаграма классів моделі			
			бази данних			
	А4	**ІАЛЦ.467200.008 Д4**	Агрегатор новин, реалізований	1		
			У вигляді web-порталу			
			Блок-схема алгоритму роботи			
			модуля-парсера			

Змн.	Арк.	№ докум.	Підп.	Дата	ІА/ЛЦ.467200.002 ОА		
Розробив		Надолінський			Агрегатор новин, реалізований у вигляді Web-порталу Опис альбому	Літ.	Аркуш 1 / Аркушів 1
Перевірив		Антонюк					
Норм. контр.		Сімоненко				НТУУ «КПІ» ФІОТ ІО-63	
Затв.		Стіренко					

ЗМІСТ

					ІА/ЛЦ.467200.003 ТЗ		
Змн.	Арк.	№ докум.	Підп.	Дата			
Розробив		Надолінський			Агрегатор новин, реалізований у вигляді Web-порталу Технічне завдання	Літ.	Аркуш 1 / Аркушів 3
Перевірив		Антонюк					
Норм. контр.		Сімоненко				НТУУ «КПІ» ФІОТ ІО-63	
Затв.		Стіренко					

1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: Агрегатор новин, реалізований у вигляді Web-порталу.

Галузь застосування: інформаційні технології.

2. ПІДСТАВА ДЛЯ РОЗРОБЛЕННЯ

Підставою для розроблення є завдання на дипломне проектування, затверджене кафедрою обчислювальної техніки Національного технічного університету України «Київський політехнічний інститут» (НТУУ «КПІ»).

3. ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для використання в якості єдиного ресурсу для моніторингу новин з різних джерел (інтернет-видань, блогів, соціальних мереж тощо), обраних користувачем індивідуально.

4. ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ

Web-ресурс повинен забезпечувати такі основні функції:

1) актуальність новинної інформації;

2) підтримку переважної більшості форматів стрічок оновлень новинних сайтів;

3) можливість динамічного оновлення вмісту web-сторінок;

4) фільтрація результатів агрегації;

5) підтримку переважної більшості форматів стрічок оновлень новинних сайтів;

6) розробку виконати з використанням веб-фреймворку (на вибір);

Додаткові вимоги:

1) наявність динамічного клієнтського інтерфейсу;

2) функціонал авторизації користувача;

3) можливість збереження результатів агрегації у базі або в якості посилання;

4) наявність зручного контекстного меню;

5) вбудована в сайт інструкція користувача (довідка).

5. ВИМОГИ ДО ПРОЕКТНОЇ ДОКУМЕНТАЦІЇ

У процесі виконання проекту повинна бути розроблена наступна документація:

1) пояснювальна записка;

2) програма та методика тестування;

3) інструкція користувача;

4) креслення:

– «Блок-схема алгоритму роботи модуля-парсера»;

– «Схема роботи програмної системи».

– «UML-діаграма класів моделі бази даних».

6. ЕТАПИ ПРОЕКТУВАННЯ

Затвердження теми роботи –	01.09.2019
Вивчення та аналіз завдання –	10.10.2019-10.12.2019
Розробка архітектури та загальної структури системи –	15.12.2019-14.01. 2019
Розробка структур окремих підсистем –	20.01. 2020-15.02. 2020
Програмна реалізація системи –	05.03. 2020-10.04. 2020
Оформлення пояснювальної записки –	11.04. 2020-20.05. 2020
Захист програмного продукту –	22.05. 2020

Пояснювальна записка
до дипломного проєкту
на тему: «Агрегатор новин, реалізований у вигляді WEB-порталу»

Київ – 2020 року

ЗМІСТ

ВСТУП

Часте відвідування багатьох різних веб-сайтів, щоб дізнатись, чи з’явилось на них щось нове, може зайняти багато часу. Технологія агрегації допомагає об'єднати багато веб-сайтів в одну сторінку, яка може відображати лише нову або свіжу інформацію з багатьох сайтів. Агрегатори скорочують час і зусилля, необхідні для регулярної перевірки веб-сайтів на наявність оновлень, при цьому дозволяють створити унікальний інформаційний простір або особисту підбірку новин. Після підписки на канал агрегатор може перевіряти наявність нового вмісту через визначені користувачем інтервали часу відповідно оновлюючи стрічку новин. На відміну від так званої підписки на push-сповіщення, агрегатор дозволяє легко скасувати підписку на канал. Стрічка часто представлена у форматах RSS або Atom, які використовують розширювану мову розмітки (XML) для структурування фрагментів інформації, які підлягають агрегуванню, при цьому результат відображається у зручному для користувача інтерфейсі. Агрегатор забезпечує компактний вигляд змісту в одному дисплеї браузера або десктопного додатку. Подібні програми пропонують переваги потенційно багатшого користувальницького інтерфейсу та можливості надання певних статей, навіть якщо комп'ютер не підключений до Інтернету. Агрегатори на базі веб-сайтів натомість є досить зручними, що дозволяє користувачам отримувати доступ до оновлених новинних каналів із будь-якого комп'ютеру, підключений до мережі. Отже, створення web-порталу для агрегації новин являє собою досить актуальну задачу.

Дана дипломна робота присвячена розробленню агрегатора новин у вигляді web-порталу, що має на меті дозволити користувачам доступ з любої платформи за наявності інтернет підключення до актуальної новинної інформації від перевіренних джерел.

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

RSS	спочатку RDF Site Summary – “RDF-резюме сайту”; пізніше з'явилися два конкуруючі підходи, Rich Site Summary – “повне резюме сайту” та Really Simple Syndication – “справді проста синдикація”.
RDF	Resource Description Framework – фреймворк опису ресурсів.
XML	Extensible Markup Language – розширювана мова розмітки.
Content Syndication	синдикація контенту – одночасне поширення інформації на різні сторінки або веб-сайти.
Content Aggregation	агрегація контенту – збір інформації з різних джерел в одному місці.
URL	Uniform Resource Locator — єдиний вказівник на ресурс.
W3C	World Wide Web Consortium – головна міжнародна організація, що розробляє й впроваджує технологічні стандарти для Всесвітнього павутиння.
IETF	Internet Engineering Task Force — відкрите міжнародне співтовариство проектувальників, учених, мережевих операторів і провайдерів, яке займається розвитком протоколів і архітектури Інтернету.
OPML	Outline Processor Markup Language – мова розмітки структури, що базується на форматі XML.
MVC	Model-view-controller – архітектурний шаблон “Модель–вигляд–контролер”.
DRY	“Don't repeat yourself” – “не повторюй себе” — принцип розробки програмного забезпечення, що направлений на уникнення дублювання інформації будь-якого вигляду.
HTTP	Hyper Text Transfer Protocol – протокол передачі гіпер-текстових документів.
ORM	Object-relational mapping – об'єктно-реляційна проекція — технологія програмування, яка зв'язує бази даних з концепціями об'єктно-орієнтованих мов програмування, створюючи «віртуальну об'єктну базу даних».
CSRF	Cross-Site Request Forgery – міжсайтова підробка запиту, тип веб атаки.
XSS	Cross Site Scripting – міжсайтовий скриптинг – тип вразливості систем у вебі.
CSS	Cascading Style Sheets – каскадні таблиці стилів.
HTML	HyperText Markup Language — мова розмітки гіпертексту.

DOM	Document Object Model – об'єктна модель документа.
AJAX	Asynchronous JavaScript And XML – асинхронний JavaScript та XML – підхід до побудови користувацьких інтерфейсів веб-застосунків.

1. АНАЛІЗ ФОРМ ПРЕДСТАВЛЕННЯ ОНОВЛЕНЬ ІНФОРМАЦІЇ НА САЙТАХ НОВИН ТА ОГЛЯД ІСНУЮЧИХ АГРЕГАТОРІВ НОВИН

1.1 . Порівняння методів представлення оновлень на новинних сайтах.

1.1.1. RSS

RSS - це відкритий метод доставки веб-контенту, який з'явився в 1995 р. [1] та регулярно змінюється. Багато веб-сайтів, веб-журналів та інших інтернет-видавців, що пропонують оновлювану інформацію, відображають вміст свого сайту як RSS-стрічку.

Користувач, який хоче отримати останні оновлення з улюблених сайтів, можете підписатися на доступні RSS-канали через застосунок, що має назву RSS-рідер. Власники сайтів з часто оновлюваним змістом (в тому числі новинні сайти) зацікавлені в створенні RSS-каналу для свого сайту.

Так як історія розвитку данної технології досить бурхлива, термін RSS в залежності від контексту може означати:

- протокол, що забезпечує відкритий метод синдикації та агрегації веб-контенту.
- стандарт для публікації регулярних оновлень веб-контенту.
- стандарт синдикації, заснований на типі XML-файлу, який знаходиться на Інтернет-сервері.
- Прикладне застосування XML, яке відповідає специфікації RDF W3C і розширюється через XML.

Відповідно те, як слід розшифровувати абревіатуру RSS залежить від версії RSS, яка використовується:

- RSS Version 0.9 - Rich Site Summary (приблизно 50% всіх RSS-стрічок використовують версію 0.91);
- RSS Version 1.0 - RDF Site Summary (приблизно 25% від загалу)

- RSS Versions 2.0, 2.0.1, and 0.9x - Really Simple Syndication (сумарно займають 25%)

RSS-канал (стрічка) - це текстовий XML-файл, який знаходиться на Інтернет-сервері. Файл RSS-каналу містить основну інформацію про сайт (заголовок, URL-адреси, опис), а також одну або кілька записів, що містять, як мінімум, заголовок, URL-адресу та короткий опис пов'язаного вмісту. Залежно від версії RSS канал має різний вигляд (різницю між різними версіями RSS в деталях буде розглянуто нижче). RSS-канали зареєстровані в реєстрі RSS, щоб зробити їх більш доступними для користувачів, зацікавлених у новинах на певну тематикою. RSS-канали можуть мати посилання на веб-сайт-джерело, що призводить до збільшення трафіку на відповідний сайт. RSS-канали оновлюються щогодини (преса та новинні збірки), деякі RSS-канали оновлюються щодня, а інші оновлюються щотижня або нерегулярно.

```
<?xml version="1.0" encoding="UTF-8" ?>
<rss version="2.0">
<channel>
 <title>RSS Title</title>
 <description>This is an example of an RSS feed</description>
 <link>http://www.example.com/main.html</link>
 <lastBuildDate>Mon, 06 Sep 2010 00:01:00 +0000 </lastBuildDate>
 <pubDate>Sun, 06 Sep 2009 16:20:00 +0000</pubDate>
 <ttl>1800</ttl>

 <item>
  <title>Example entry</title>
  <description>Here is some text containing an interesting description.</description>
  <link>http://www.example.com/blog/post/1</link>
  <guid isPermaLink="false">7bd204c6-1655-4c27-aeee-53f933c5395f</guid>
  <pubDate>Sun, 06 Sep 2009 16:20:00 +0000</pubDate>
 </item>

</channel>
</rss>
```

Рис. 1.1 – приклад стрічки в форматі RSS 2.0

Переваги RSS для користувачів – підписників на оновлення:

- ✔ усі новини в одному місці: існує можливість підписатися на кілька груп новин, а потім налаштувати RSS-рідер таким чином, щоб усі новини були на одній сторінці. Це дає змогу заощадити багато часу, який інакше був би витрачений на відкриття декількох сайтів з новинами.
- ✔ новини, коли в них є потреба: замість того, щоб хаотично переглядати новинні веб-сайти очікуючи оновлень та гаючи час, користувачі RSS-рідерів можуть просто перейти до свого читача RSS, тоді, коли існує необхідність прочитати новини. Крім того, через відсутність "всього зайвого" статті з RSS-каналів завантажуються швидше, ніж аналогічна інформація на веб-сайтах, і можуть бути доступні для читання у режимі режимі офлайн.
- ✔ отримання тільки потрібних новини: новини з RSS-каналу мають вигляд заголовків та короткого опису для того, щоб користівачі могли легко проглядати суть та переходити лише на ті посилання з історіями, які їх зацікавили.
- ✔ свобода від перевантаження електронної пошти: користувачі не отримують жодного електронного листа з будь-якими новинами чи оновленнями блогу. Достатньо просто перейти до свого агрегатора, і ви знайдете оновлені новини чи блог автоматично, в момент зміни зміна на RSS-сервері.
- ✔ легкість поширення інформації та повторна публікація: користувач можете бути як підписником, так і публікатором. Наприклад, у такої людини може бути веб-сайт, який збирає новини з різних інших сайтів (вручну або автоматично), а потім публікує їх. RSS дозволяє легко фіксувати ці новини та відображати їх на своєму сайті.

Переваги RSS для власників сайтів – тих, хто публікує новини:

- ✔ простіша публікація: RSS - це дійсно проста публікація. Вам не потрібно підтримувати базу даних підписників, щоб надсилати їм свою інформацію, натомість вони отримуватимуть доступ до вашого каналу за допомогою агрегатора чи RSS-рідера і автоматично отримуватимуть оновлений контент.
- ✔ більш простий процес написання: Якщо у вас є нова інформація на вашому веб-сайті, вам потрібно лише написати RSS-канал у формі заголовків та коротких описів і додати до цього гіперпосилання на першоджерело – ваш сайт.
- ✔ покращені стосунки з вашими підписниками: оскільки люди підписуються зі своєї сторони, вони не відчувають себе так, ніби ви нав'язуєте їм свій контент – відповідно збільшується ступінь довіри до вашого сайту.
- ✔ забезпечення доступу до ваших підписників: RSS може виконувати роль спам-фільтру, так як ваші підписники отримують канали, на які вони підписалися, і більше нічого.
- ✔ посилання на ваш сайт: RSS-канали завжди містять посилання на веб-сайт, що збільшує об'єм трафіку на нього.
- ✔ актуальність та своєчасність: Ваші підписники завжди мають найновішу інформацію з вашого сайту.

Різниця між версіями RSS проявляється в наступному:

RSS v0.91 Feed Format: RSS v0.91 спочатку був випущений Netscape у 1999 році. У нього немає заголовка RDF. RSS v0.91 має функції сценарію RSS-версії Дейва Вайнера News 2.0b1. RSS v0.91 підтримує міжнародні мови та кодування, також підтримує визначення висоти та ширини зображення та текст опису заголовків.

RSS v1.0 Feed Format: RSS 1.0 - єдина версія, розроблена з використанням стандарту W3C RDF (Resource Description Framework). RSS 0.91 та RSS 2.0 легше зрозуміти, ніж RSS 1.0.

RSS v2.0/2.01 Feed Format: RSS 2.0 / 2.01 дуже схожий на RSS 0.9x. RSS 2.0 / 2.01 додає модулі простору імен та шість додаткових елементів до RSS 0.9x.

RSS аж ніяк не є ідеальним форматом, але він дуже популярний і широко підтримується. Фіксована остаточна специфікація - це те, що RSS потрібно давно. Однак специфікація RSS для всіх практичних цілей заморожена у версії 2.0.1. Але існує вірогідність появи версій 2.0.2 або 2.0.3. Але лише з метою уточнення специфікації, а не для додавання нових функцій у формат. Подальша робота повинна відбуватися в модулях, використовуючи простори імен, і в абсолютно нових форматах синдикації, з новими іменами.

1.1.2. Atom

Atom - це назва формату синдикації метаданих і веб-контенту на основі XML та протоколу на рівні програми для публікації та редагування веб-ресурсів, що належать до періодично оновлюваних веб-сайтів.

Формат Atom був розроблений як альтернатива RSS. Бен Тротт, прихильник нового формату, який згодом отримав назву Atom, вважав, що RSS має обмеження та недоліки - такі як відсутність постійних інновацій та необхідність залишатися назад сумісними - і що у нового дизайну є переваги [2].

Прихильники нового формату сформували робочу групу IETF Atom Publishing Format and Protocol. Формат синдикації Atom був опублікований як запропонований IETF стандарт у RFC 4287 (грудень 2005 р.), А протокол публікації Atom був опублікований як RFC 5023 (жовтень 2007 р.).

На сьогоднішній день люди використовують Atom та інші формати веб-синдикації для багатьох цілей, включаючи журналістику, маркетинг, повідомлення про помилки або будь-яку іншу діяльність, що включає періодичні оновлення чи публікації. Atom також пропонує стандартний спосіб експорту цілого блогу або його частин для резервного копіювання або імпорту в інші системи блогів.

Деякі веб-сайти дозволяють людям вибирати між RSS-або Atom-форматами веб-каналів; інші пропонують лише RSS або лише Atom. Зокрема, багато веб-сайтів в форматі блогу та вікі пропонують свої веб-канали оновлень у форматі Atom.

Коли Atom з'явився як формат, призначений для суперництва або заміни RSS, CNET описав мотивацію своїх творців так: "Опоненти Winer шукають нового формату, який би роз'яснив неоднозначності RSS, консолідував його декілька версій, розширив його можливості та потрапив під під егідою традиційної організації з стандартизації ". [3]

```xml
<?xml version="1.0" encoding="utf-8"?>

<feed xmlns="http://www.w3.org/2005/Atom">

    <title>Example Feed</title>
    <subtitle>A subtitle.</subtitle>
    <link href="http://example.org/feed/" rel="self" />
    <link href="http://example.org/" />
    <id>urn:uuid:60a76c80-d399-11d9-b91C-0003939e0af6</id>
    <updated>2003-12-13T18:30:02Z</updated>


    <entry>
        <title>Atom-Powered Robots Run Amok</title>
        <link href="http://example.org/2003/12/13/atom03" />
        <link rel="alternate" type="text/html" href="http://example.org/2003/12/13/atom03.html"/>
        <link rel="edit" href="http://example.org/2003/12/13/atom03/edit"/>
        <id>urn:uuid:1225c695-cfb8-4ebb-aaaa-80da344efa6a</id>
        <updated>2003-12-13T18:30:02Z</updated>
        <summary>Some text.</summary>
        <content type="xhtml">
            <div xmlns="http://www.w3.org/1999/xhtml">
                <p>This is the entry content.</p>
            </div>
        </content>
        <author>
            <name>John Doe</name>
            <email>johndoe@example.com</email>
        </author>
    </entry>

</feed>
```

Рис. 1.2 – приклад стрічки в форматі Atom 1.0

Тім Брей, який зіграв головну роль у створенні Atom, дав короткий опис деяких способів, якими Atom 1.0 відрізняється від RSS 2.0: [4]

- формати дати – специфікація RSS 2.0 покладається на використання часових позначок у форматі RFC 822 для передачі інформації про те, коли елементи в каналі були створені та останній раз оновлені. Робоча група Atom замість цього використала часові позначки, відформатовані відповідно до правил, визначених RFC 3339 (що є підмножиною ISO 8601; для відмінностей див. Додаток А до RFC 3339).
- інтернаціоналізація – незважаючи на те, що у словнику RSS є механізм вказівки людської мови на канал, немає можливості вказати мову для окремих елементів або елементів тексту. Atom, з іншого боку, використовує стандартний атрибут *xml: lang*, щоб можна було вказати мовний контекст для кожного фрагмента вмісту, читаного людиною, у каналі. Atom також відрізняється від RSS тим, що він підтримує використання інтернаціоналізованих ідентифікаторів ресурсів, які дозволяють посиланням на ресурси та унікальні ідентифікатори містити символи, що знаходяться за межами набору символів ASCII США.
- модульність – елементи словника RSS зазвичай не використовуються в інших XML-лексиках. Синтаксис Atom був спеціально розроблений, щоб дозволити повторне використання елементів за межами документа подачі Atom. Наприклад, не рідкість знайти елементи *atom: link*, що використовуються в RSS-каналах RSS 2.0.

Таблиця 1.1 – різниця ключових слів форматів RSS 2.0 та Atom 1.0 (* означає, що елемент є обов'язковим)

RSS 2.0	Atom 1.0
author	author *
category	category
channel	feed
copyright	rights
—	subtitle
description *	summary and/or content
generator	generator
guid	id *
image	logo
item	entry
lastBuildDate (in channel)	updated *
link *	link *
managingEditor	author or contributor
pubDate	published (subelement of entry)
title *	title *
ttl	—

Перешкоди для прийняття:

Незважаючи на появу Atom як стандарту, що пропонується IETF, і рішення великих компаній, таких як Google, прийняти Atom, використання старих і більш відомих форматів RSS продовжується. Для цього є кілька причин:

1) підтримка додатків у RSS 2.0 призвела безпосередньо до розвитку подкастингу. Хоча багато додатків для подкастингу, такі як iTunes, підтримують використання Atom 1.0, RSS 2.0 залишається кращим форматом.
2) багато сайтів вирішують публікувати свої канали лише в одному форматі. Наприклад, CNN та The New York Times пропонують свої веб-канали лише у форматі RSS 2.0.

Статті з новин про канали веб-синдикації все частіше використовують термін "RSS" для загального позначення будь-якого з декількох варіантів формату RSS, таких як RSS 2.0 та RSS 1.0, а також формату Atom. [16]

1.2. Приклади агрегаторів новин.

1.2.1. Google News

Google News - це додаток для агрегування новин, розроблене Google. У ньому представлено безперервний, що настроюється потік статей, організованих від тисяч видавців та журналів. Новини Google доступні як додаток для Android, iOS та Інтернету.

Google випустила бета-версію у вересні 2002 року, а офіційну програму - у січні 2006 року. Початкова ідея була розроблена Крішна Бхаратом.

Служба була описана як найбільший у світі агрегатор новин[17].

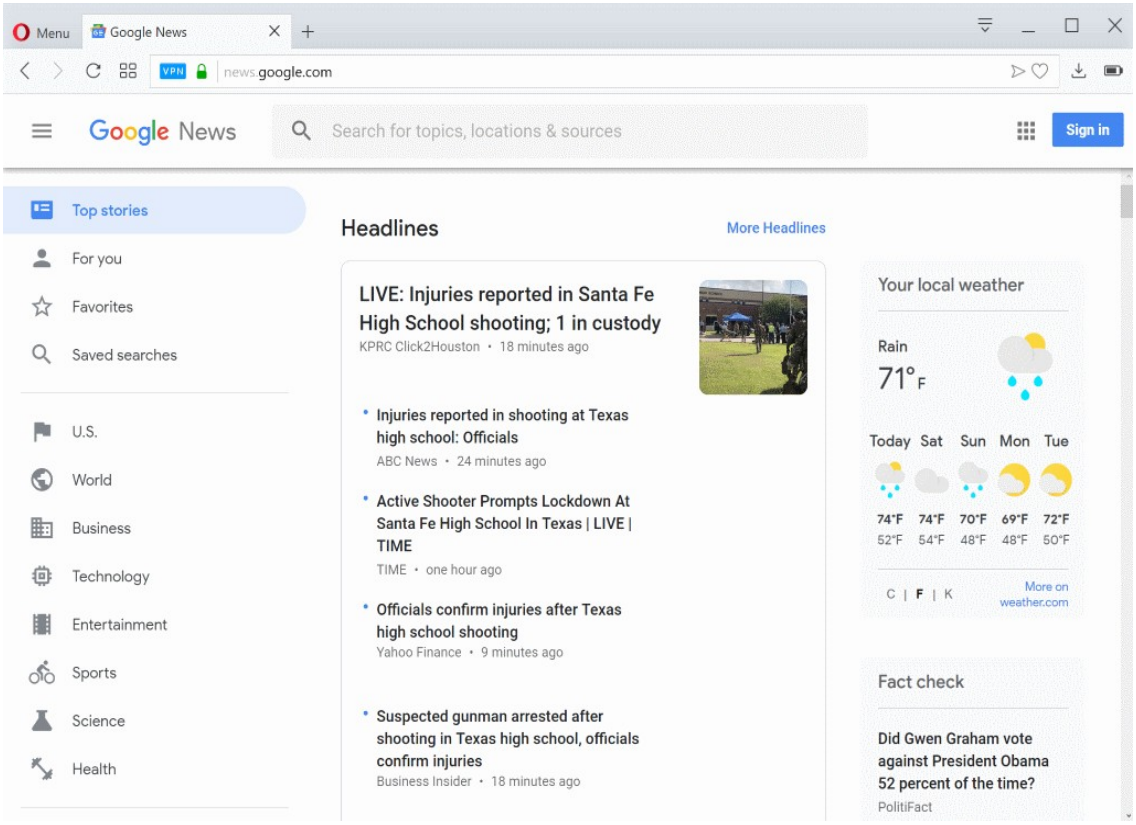


Рис. 1.3 – головна сторінка Google News.

Особливості:

Спадне меню у верхній частині результатів пошуку дає змогу користувачам вказати період часу, у який вони хочуть шукати статті. Це меню включає такі параметри, як: минулий день, минулий тиждень, минулий місяць або спеціальний діапазон.

Користувачі можуть запитувати *"сповіщення"* по електронній пошті на різні теми ключових слів, підписавшись на Google News Alerts. Електронні листи надсилаються підписникам щоразу, коли в Інтернеті надходять статті новин, що відповідають їх запитам. Сповіщення також доступні через RSS та Atom.

Користувачі мали змогу *налаштувати відображені розділи*, їхнє розташування на сторінці та кількість історій, що відображаються за допомогою інтерфейсу перетягування на основі JavaScript. Однак для американського сайту це було відключено на користь нового макета; розгортання цього плану планується найближчим часом для інших

місцевостей. Історії з різних видань Новин Google можна об'єднати, щоб утворити одну персоналізовану сторінку, з опціями, що зберігаються у файлі cookie. Послуга інтегрована в історію пошуку Google з листопада 2005 року. Після її закінчення з бета-версії було додано розділ, в якому відображаються рекомендовані новини на основі історії пошуку користувачів Google News та статті, на які користувач натискав (якщо користувач підписався для історії пошуку).

У травні 2018 року була представлена оновлена версія Новин Google, яка включала функції *штучного інтелекту*, щоб допомогти користувачам знайти відповідну інформацію.[18]

1.2.2. The Old Reader

The Old Reader - це веб-агрегатор новин, який доставляє веб-сайт, блог та інший Інтернет-вміст у поштову скриньку. Служба зародилася, коли Google видалив соціальні функції з Google Reader; сайт підтримує обмін соціальними мережами, включаючи можливість "сподобатися" вмісту та знаходити друзів через соціальні мережі.

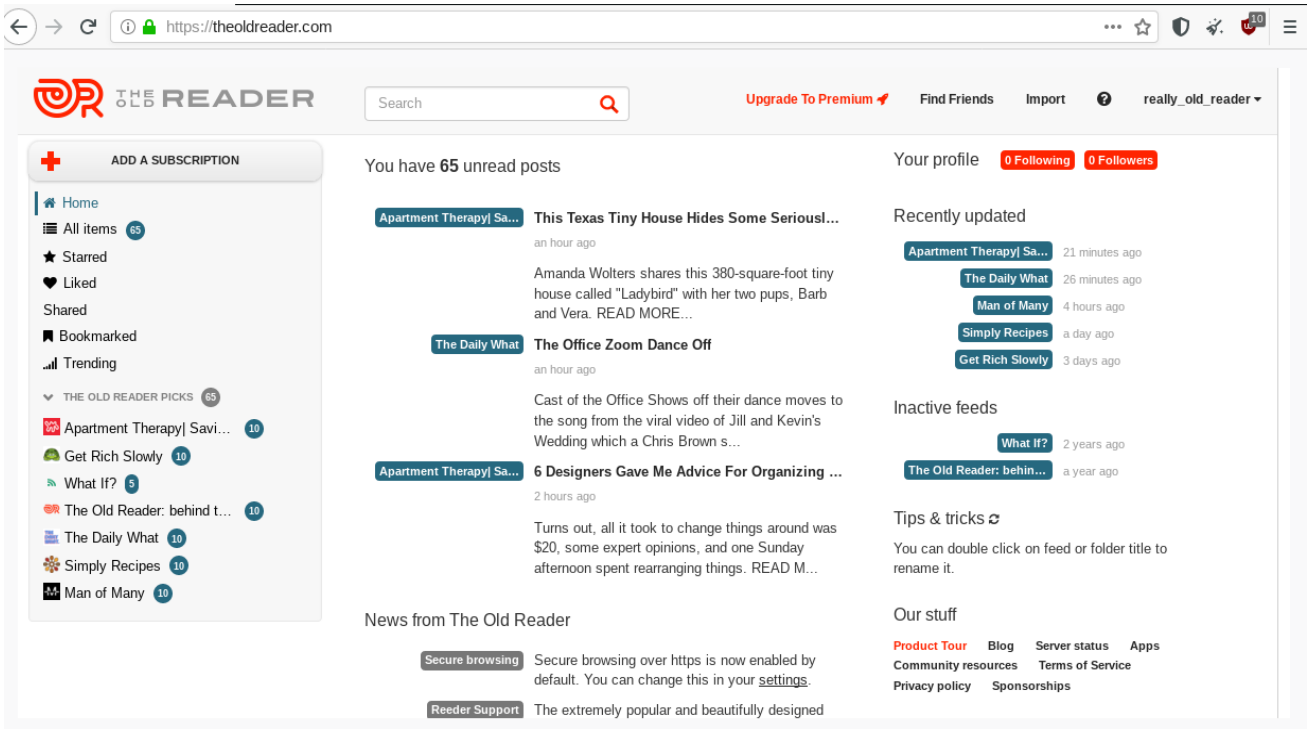


Рис. 1.4 – головна сторінка The Old Reader.

Особливості:

Old Reader *безкоштовний до 100 каналів* і пропонує *преміум-версію* з повнотекстовим пошуком і до 500 підписок та 6 місяців зберігання після публікації. Колишні користувачі Google Reader або інших читачів RSS можуть імпортувати канали за допомогою *експорту OPML. Закладка браузера* дозволяє користувачам надсилати веб-сторінки безпосередньо до облікового запису The Old Reader[19].

Служба інтегрована з Facebook або Google, щоб допомогти користувачам знайти друзів, які також користуються сайтом. Існує також підтримка читабельності, Instapaper та Spritz, що допомагає швидше читати вміст.

Організація каналів у папки та категорії – створити нову папку каналу так само просто, як перетягнути канал на порожнє місце під останньою папкою. Порожні папки видаляються автоматично. Перейменувати папку за замовчуванням ("Subscriptions") неможливо, але можно перемістити всі свої канали з неї в інші папки, і вона повністю зникне.

Існує *навігаційне меню,* що дозволяє одним натисканням кнопки перейти до наступної чи попередньої публікації. Також є можливість переключитися на "List view" режим, який згортає всі публікації, дозволяє переглядати заголовки та розширювати лише ті статті, які вас цікавлять.

Підтримка використання *комбінацій клавіш* для навігації.

Під кожним повідомленням є *кнопки дій.* Якщо ви натиснете "Like", ваше ім'я з'явиться в списку користувачів, яким сподобалось це повідомлення. Публікація також перейде до вашої папки "Liked".

Обговорення постів – існує вкладка "Find Friends", для відображення тих, хто з ваших друзів у Facebook чи у контактах Google використовує "The Old Reader". Існує можливість *шукати друзів* по імені.

Усі публікації, якими ви поділилися, знаходяться у папці "Shared". Елементи, якими поділилися ваші друзі, знаходяться у папці "Following".

Як завжди, користувачі можуть прочитати всю папку або вибрати певне ім'я користувача, щоб побачити всі публікації, якими користувався цей користувач.

Користувачі можуть коментувати будь-яку публікацію, якою поділились їх друзі за допомогою кнопки «Comment».

1.2.3. Feedly

Feedly – це додаток для збирання новин для різних веб-браузерів та мобільних пристроїв, на яких працює iOS та Android. Він також доступний як хмарний сервіс. Він збирає стрічки новин з різних Інтернет-джерел, щоб користувач міг налаштувати та поділитися з іншими. Feedly вийшов у реліз у 2008 році.

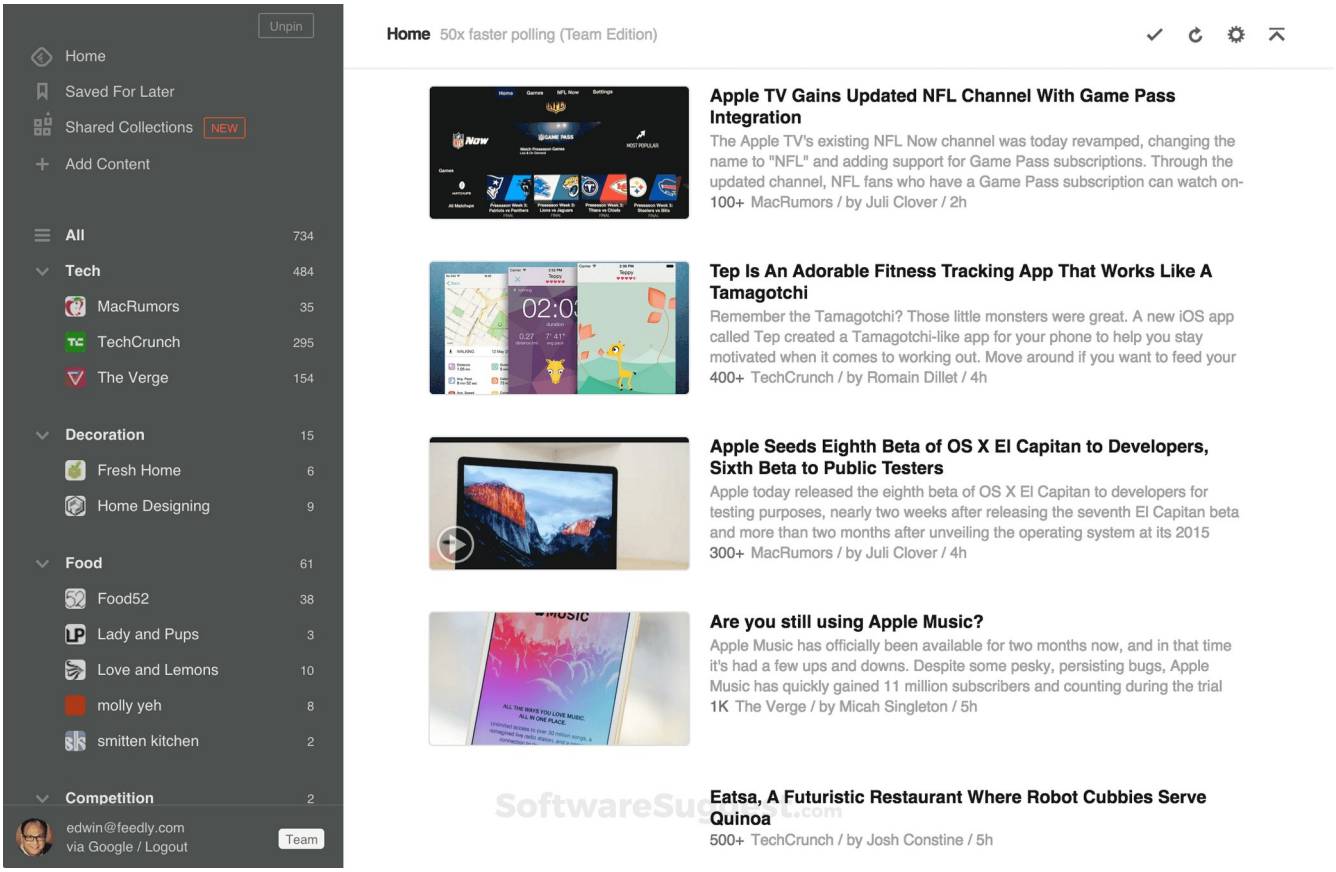


Рис. 1.5 – домашня сторінка feedly.

Особливості:

Налаштування та навігація – вхідна сторінка Feedly має мінімалістичний, липово-зелений дизайн, у якому логотипи різних публікацій вирівняні внизу інтерфейсу. Для входу в систему потрібно використовувати обліковий запис Google, а це означає, що якщо у вас цього немає, вам потрібно буде створити його. Це може відвернути деяких потенційних користувачів Feedly, але якщо ви вже користувались Google Reader, ви відчуєте себе як вдома.

Щойно ви надаєте Feedly право доступу до свого облікового запису Google, він швидко завантажує вашу домашню сторінку. Він складається з трьох розділів: Основний розділ із вмістом (у якому розміщені Пропоновані історії), стовпець зліва від нього (який демонструє розділи для свіжих історій, збережених історій та інструмент для додавання нових джерел) та бічну панель праворуч про нього (який відображає ваші канали та кілька рекламних оголошень Amazon)[15]. Макет є базовим і простий у навігації. У Feedly є кілька тем і переглядів категорій для подальшого налаштування.

Функціонал перегляду стрічки наступний: рекомендовані історії супроводжуються мініатюрними зображеннями та посиланнями, які дозволяють швидко зберегти статтю або позначити її як прочитану. Збережені статті залишаються в основній області вмісту, але ті, що позначені як прочитані, одразу зникають до категорії "Збережені" і замінюються новими історіями, коли вони стають доступними що є досить приємним. Рекомендовані історії за замовчуванням - це статті, опубліковані в той день, коли був отриманий доступ до каналу. Також можно позначати теги за допомогою ключових слів, які діють як категорії, що можна натискати, що дозволяє швидко бачити, наприклад, "політика".

Додавання нових джерел є доволі простим. Натиснувши "Додати веб-сайт", відкриється вікно, в якому відображаються такі категорії, як "Кулінарія", "Бізнес" та "Ігри". При натисканні на категорію

відображаються численні пов’язані джерела, які ви або додаєте до своїх підписок окремо, або як пакет. Це прекрасний спосіб відкрити новий вміст. Додати контент у Feedly можна шукаючи також заголовок джерела або ввівши його URL-адресу[15].

Якщо натиснути історію, ви відкриєте вміст цієї сторінки у Feedly, і залежно від того, як у джерелі встановлено RSS-канал, ви знайдете або уривки історії, або повні статті та соціальні медіа.

Взаємодія з соціальними мережами – Feedly також працює з соціальними медіа, щоб продемонструвати посилання, які надаються у ваших каналах[15]. Після налаштування облікових даних у Facebook та Twitter, Feedly відображає спільні посилання друзів у правій колонці. Це дозволяє користувачам побачити всі цікаві посилання в одному зручному місці.

ВИСНОВКИ ДО РОЗДІЛУ 1

Огляд форм представлення оновлень, які найбільш поширені в мережі, зокрема на новинних сайтах, а також блогах, каналах та різноманітних сайтах з синдикацією новин, показав, що існує два досить подібні по структурним показникам формати, а саме RSS (використовуються різні версії) та Atom. Причому подальше ознайомлення з популярними web-агрегаторами новин показало, що працювати з даними форматами можуть всі з них.

В цілому в ході вивчення можливостей, що пропонуються даними агрегаторами можна виокремити наступні типові особливості:

- система авторизації користувача з її відповідними перевагами – наприклад створенням власних підбірок новин з обраних джерел, що зберігаються в базі данних та використання в повній мірі всіляких соціальних можливостей;
- зручний інтерфейс новинних стрічок;
- можливість вибору способу відображення агрегованих статей або на сайті агрегатора, або перехід на сайт-джерело за бажанням (для зменшення втрат трафіку користувачів на сайті-джерелі);
- гнучкі фільтри агрегованого контенту тощо.

Логічно припустити, що реалізація цих особливостей має бути пріорітетною ціллю на етапах проектування та власне розробки готового рішення.

2. ОБГРУНТОВАНИЙ ВИКЛАД РІШЕННЯ ПРОБЛЕМИ АГРЕГАЦІЇ НОВИННОЇ ІНФОРМАЦІЇ З РІЗНИХ ДЖЕРЕЛ

Під час аналізу існуючих рішень в області агрегаторів новинного контенту стало зрозуміло, що найбільш прийнятним, компромісним рішенням, як для потенційних користувачів, так і для розробника буде саме розробка агрегатора новин саме в якості WEB-порталу. Подібних сайтів існує велика кількість і причини цього криються в перевагах WEB розробки, а саме:

- доступність на усіх платформах при наявності звичайного інтернет підключення;
- легкість доступу до сайту – користувачу достатньо лише ввести URL, не потрібно нічого завантажувати та встановлювати на свій пристрій;
- постійно найсвіжіша версія застосунку з найновішим функціоналом, знову ж таки з боку користувача ніяких зусиль для цього докладати не треба;
- швидкість та якість розробки – новітні фреймворки вирішують частину тривіальних проблем веб-розробки “з коробки”, програмісту ж лишається найскладніше – продумати архітектуру, бізнес-логіку, дизайн та налаштувати взаємодію компонентів.

Організацією архітектури краще за все описати за допомогою одного з програмних шаблонів, основною метою яких є можливість гнучкої розробки компонентів та зручність подальшої підтримки роботи застосунку, наприклад легка зміна однієї зі структурних одиниць програми, перехід на нову технологію тощо. Архітектурні шаблони дуже влучно описує відома історична фраза “Розділяй та володарюй”. Одним із таких архітектурних шаблонів є *MVC* (*Model View Controller*).

2.1 Опис архітектурного шаблону MVC

Модель-вид-контролер (або Модель-вигляд-контролер, англ. *Model-view-controller, MVC*) – це паттерн дизайну програмного забезпечення, який зазвичай використовується для розробки інтерфейсів користувача та розділяє відповідну логіку програми на три взаємопов'язані елементи. Це робиться для відокремлення внутрішніх уявлень інформації від способів подання та прийняття інформації від користувача. Традиційно використовувалась для графічних інтерфейсів користувачів настільних ПК (GUI), пізніше ця модель стала популярною для розробки веб-додатків.

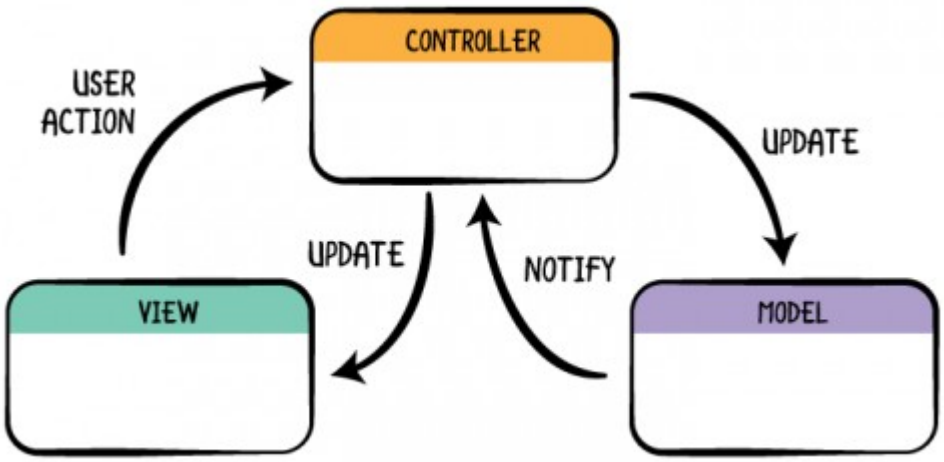


Рис. 2.1 – схематичне зображення взаємодії компонентів MVC.

Характеристика окремих компонентів:

- Модель: центральний компонент шаблону. Це динамічна структура даних програми, незалежна від інтерфейсу користувача. Безпосередньо керує даними, логікою та правилами програми.
- Вид: будь-яке представлення інформації, наприклад діаграми, слайди або таблиці. Можливі кілька переглядів однієї й тієї

самої інформації, такі як гістограма для управління та таблична таблиця для бухгалтерів.

- ◆ Контролер: приймає введення данних та перетворює його в команди для моделі чи виду.

На додаток до поділу програми на ці компоненти, шаблон MVC визначає і взаємодію між ними.

- Модель відповідає за управління даними програми. Вона отримує введені дані користувача від контролера.
- Вид являє собою представлення моделі у певному форматі.
- Контролер відповідає на введення користувача та здійснює взаємодію з об'єктами моделі даних. Контролер отримує вхідні дані, додатково перевіряє їх, а потім передає моделі.

Як і в інших моделях програмного забезпечення, MVC виражає «ядро рішення» проблеми, дозволяючи пристосувати її до кожної системи. Конкретні конструкції MVC можуть суттєво відрізнятися від традиційного опису.

Переваги MVC:

- ✔ Одночасний розробка - кілька програмістів можуть одночасно працювати над моделлю, контролером та видами.
- ✔ Висока згуртованість - MVC дозволяє логічно групувати пов'язані дії на контролері разом. Види конкретної модель також групуються.
- ✔ Мала кількість зв’язків – властивість, що витікає з означення MVC така, що між моделями, видами чи контролерами немає необхідності в великій зв’язності.
- ✔ Простота модифікації - завдяки поділу обов'язків між компонентами шаблону майбутнярозробка чи модифікація стають значно простішими та зручнішими.

- ✔ Кілька представлень для моделі - моделі можуть мати кілька видів для перегляду в різних ситуаціях.
- ✔ Зручність тестування - за чіткішого розмежування проблем кожну частину можна краще перевірити самостійно (наприклад, виправляючи модель при цьому не змінюючи її вид).

Недоліки MVC:

- ✗ Керованість кодом - навігація за рамками може бути іажчою, оскільки вона вводить нові складнощі та вимагає від користувачів адаптації до критеріїв декомпозиції MVC.
- ✗ Загальний вигляд ознак, розбитих на частини - розкладання програмних функцій на три взаємопоєднані частини викликає розсіювання. Таким чином, від розробників вимагається підтримувати послідовність та корректність декількох представлень одночасно.
- ✗ Підверженість неминучій кластеризації - програми, як правило, мають сильну взаємодію між тим, що бачить користувач, і тим, що користувач використовує. Тому обчислення та стан кожної функції мають тенденцію кластеризуватися в одну з 3 програмних частин, ставлячи під сумнів переваги MVC.
- ✗ Надмірність кодової бази - через те, що обчислення та стан програми зазвичай кластеризовані в одну з 3-х частин, інші частини вироджуються або в придатки, або в фоновий код, що існує лише для задоволення схеми MVC.
- ✗ Виражена крива навчання - знання про декілька технологій стає нормою. Розробники, що використовують MVC, мають бути кваліфікованими у кількох технологіях.
- ✗ Відсутність додаткової вигоди - програми користувальницького інтерфейсу вже розбиваються на

компоненти, а повторне використання коду та його незалежность досягається завдяки архітектурі компонентів, що не залишає ніякої додаткової вигоди від використання MVC.

2.2 Схема та структура пропонованого рішення проблеми розробки WEB-порталу агрегації новин

Для створення WEB-сайту з використанням напрацювань архітектурного шаблону MVC необхідна Модель (буде представлена базою данних та сутностями-класами), Вид (буде представлений клієнтською частиною — так званим front-end-ом) та Контролером (логікою взаємодії між Видом та Моделлю). Окрім цього, так як існує задача агрегації контенту з різних мережевих джерел в режимі постійних оновлень, буде необхідний окремий модуль-парсер (в ідеалі також щоб його робота могла виконуватися паралельно або одночасно було запущено декілька екземплярів). В такому разі Контролер повинен так чи інакше здійснювати звернення до данного модулю.

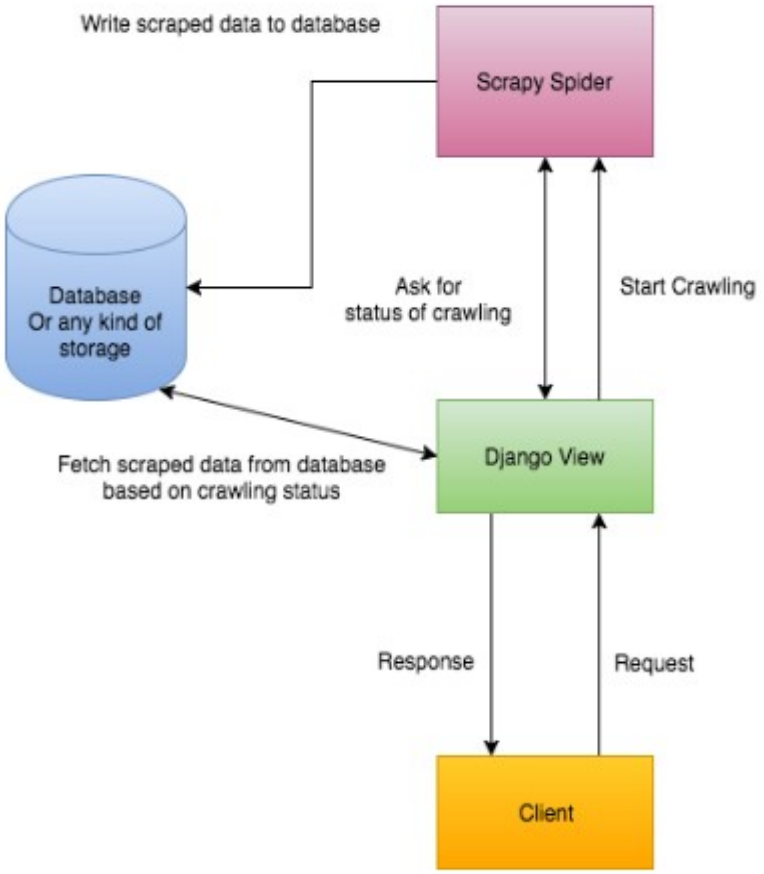


Рис. 2.2 — загальна схема пропонованого рішення.

Приблизний алгоритм роботи застосунку наступний:

1. Клієнт (Вид) надсилає запит з URL-адресою (-ами) сайтів для сканування на предмет оновлень.
2. Контролер запускає парсер, що запустити екземпляр модуля, щоб сканувати цю URL-адресу.
3. Контролер повинен повідомити Клієнту, що сканування тільки почалося і відобразити статус очікування.
4. Модуль-парсер завершує сканування і зберігає витягнуті дані в базу даних.
5. Контролер отримує ці дані з бази даних і повертає їх Клієнту.
6. В залежності від Виду дані будуть представлені користувачу.

2.3 Реалізація шаблону MVC

Для написання функціоналу, необхідного за вказівками архітектурного паттерну Модель-Вид-Контролер була вибрана мова Python3 та популярний безкоштовний з відкритим кодом WEB-фреймворк Django, який відповідає архітектурній схемі шаблону (MTV — Model-Template-View *укр. Модель-Шаблон-Вид,* по-суті аналогічний MVC), написаний на данній мові програмування. Вибір обгрунтований тим, що основна мета Django - полегшити створення складних веб-сайтів, керованих базами даних. Окрім цього, фреймворк наголошує на повторному використанні та «вбудованості» компонентів, їх низькій зв'язанності, меншій кількості коду, швидкій розробці та принципі DRY[20], що в перекладі з англійської мови означає "не повторюй себе". Python в данному випадку використовується всюди, навіть для файлів налаштувань та моделей даних. Django також надає додатковий адміністративний інтерфейс для створення, читання, оновлення та видалення, який динамічно генерується за допомогою інтроспекції — можливості визначити тип і структуру об'єкта під час виконання програми, — та налаштовується через адміністраторські моделі.

Корисні вбудовані компоненти Django:

Незважаючи на наявність власної номенклатури, як-от іменування "Видами" ("Views") об'єктів, що викликаються та відповідають за генерацію HTTP-відповіді, основна структура Django може розглядатися як архітектура MVC. [6] Він складається з об'єктно-реляційного відображення (ORM, *англ.* "object-relational mapper"), який здійснює посередництво між моделями даних (визначеними класами Python) та реляційною базою даних ("Модель"), системою обробки HTTP-запитів із системою веб-шаблонів ("Вид") та диспетчер URL-адрес на базі регулярних виразів ("Контролер").

До основної бази також входять[6]:

- легкий і окремий веб-сервер для розробки та тестування;
- система серіалізації та перевірки форм, яка здійснює перетворення форм HTML у значення, придатні для зберігання в базі даних;
- шаблонна система, яка використовує поняття успадкування, запозичене з об'єктно-орієнтованого програмування;
- фреймворк кешування, який може використовувати будь-який із декількох методів кешування;
- підтримка класів проміжного програмного забезпечення, яке можуть втручатися на різних етапах обробки запитів та виконувати спеціальні функції;
- внутрішня система диспетчера, яка дозволяє компонентам програми передавати події один одному заздалегідь визначеними сигналами;
- система інтернаціоналізації, що включає переклади власних компонентів Django на різні мови;
- система серіалізації, яка може виробляти та читати представлення XML та / або JSON примірників моделі Django;
- система для розширення можливостей власного шаблонного двигуна Jinja;
- інтерфейс до вбудованого тестового модуля *unit test* Python;
- фреймворк Django REST — потужний і гнучкий інструментарій для створення веб-API.

На додачу до цього основний дистрибутив Django також поєднує ряд програм у своєму пакеті "contrib", до яких входять[6]:

- розширювана система аутентифікації;
- динамічний адміністративний інтерфейс;
- інструменти для генерації каналів синхронізації RSS та Atom;

- фреймворк "Sites", який дозволяє одній установці Django запускати кілька веб-сайтів разом, кожен з яких може мати власний вміст та застосунки;
- інструменти для створення Google Sitemap — XML-файлу, у якому перераховані URL-адреси веб-сайту. Це дозволяє веб-розробникам включати додаткову інформацію про кожну URL-адресу: коли вона востаннє оновлювалася, як часто вона змінюється та наскільки вона важлива стосовно інших URL-адрес сайту;
- вбудований захист від міжсайтової підробки запиту (CSRF), міжсайтового скриптингу (XSS), SQL-ін'єкцій, взламування паролів та інших типових веб-атак, при цьому більшість захисних мір увімкнено за замовчуванням

2.4 Реалізація модуля парсера оновлень веб-сайтів.

Оскільки серверна частина застосунку написана на мові Python, яка славиться рясною кількістю бібліотек та модулів в тому числі й тих, що допомагають вирішити задачу парсингу інтернет-сторінок, логічно використати одну з них в якості фундаменту для реалізації власного модуля. Як результат був вибраний *Scrapy* — вільний WEB-фреймворк з відкритим кодом, написана на Python. Спочатку розроблений для парсингу веб-сторінок, він також може використовуватися для вилучення даних за допомогою API або як веб-сканер загального призначення[5].

Архітектура проекту Scrapy побудована навколо "павуків", які є автономними сканерами, яким надається набір інструкцій. Даний фреймворк також побудований за принципом DRY, аналогічно до Django. Це, в свою чергу, спрощує створення та масштабність великих скануючих проектів, дозволяючи розробникам повторно використовувати їх код. Scrapy також забезпечує оболонку командного

рядка для веб-сканування, яку розробники можуть використовувати для перевірки своїх припущень щодо поведінки сайту.

Одна з головних переваг Scrapy полягає в тому, що запити плануються та обробляються асинхронно. Це означає, що Scrapy не потрібно чекати, коли запит буде закінчений і оброблений, він може надіслати ще один запит або тим часом робити інші речі. Іншими словами, запити можуть продовжуватися, навіть якщо деякі з них обробити не вдається або виникає помилка під час обробки. Це робить можливим дуже швидкий та безвідмовний парсинг (надсилаючи кілька незалежних один від одного запитів в один момент)[13].

Scrapy також забезпечує контроль над ввічливістю сканування за допомогою декількох налаштувань. Можливо виконувати такі дії, як встановлення затримки завантаження між кожним запитом, обмеження кількості одночасних запитів на домен або на IP-адресу, і навіть використовування розширення з автоматичним обмеженням, що має змогу виставляти подібні параметри базуючись на поточному завантаженні сервера.

Scrapy надає безліч потужних функцій для полегшення та ефективності парсингу, таких як:

- вбудована підтримка для вибору і отримання даних з джерел HTML / XML за допомогою розширеного набору CSS селекторів і виразів XPath, з допоміжними методами для вилучення за допомогою регулярних виразів;
- інтерактивна консоль оболонки (відома як IPython) для випробування виразів CSS та XPath для парсингу даних, дуже корисна при написанні або налагодженні так званих павуків — парсерів;
- вбудована підтримка для генерації експорту стрічок у декількох форматах (JSON, CSV, XML) та зберігання їх у декількох пакетах (FTP, S3, локальна файлова система);

- надійна підтримка кодування та автоматичне виявлення для роботи з іноземними, нестандартними та несправними деклараціями кодування;
- сильна підтримка розширюваності, що дозволяє вам підключати власну функціональність за допомогою сигналів та чітко визначеного API;
- широкий вибір вбудованих розширень та програмного забезпечення проміжного рівня для обробки:
  - файли cookie та обробка сеансів;
  - HTTP-функції, такі як стиснення, аутентифікація, кешування;
  - підміна User-Agent параметру;
  - робота з файлом robots.txt;
  - обмеження глибини сканування;
  - та інші.
- телнет консоль для підключення в консоль Python, яка працює всередині процесу Scrapy, для аналізу та процесу відладки;
- плюс інші особливості, такі як павуки для багаторазового використання для сканування сайтів із відображення сайту та каналів XML / CSV, медіаконвеєр для автоматичного завантаження зображень (або будь-якого іншого виду контенту), пов'язаних із розпаршеними елементами, кешування DNS-роздільника тощо.

Для зв'язку між модулем-парсером та Контролером Django буде використовуватися *scrapyd* — додаток (як правило, запускається у фоновому режимі на окремому порту сервера), який прослуховує запити для запуску павуків і породжує процес для кожного з них, що по-суті виконує запуск scrapy-парсера звичайним методом (можлива також передача опціональних параметрів).

Scrapyd також виконує паралельно декілька процесів, розподіляючи їх у фіксованій кількості слотів, заданих параметрами max_proc та max_proc_per_cpu, запускаючи якомога більше процесів для обробки навантаження.

Окрім диспетчеризації та управління процесами, Scrapyd надає веб-сервіс JSON для завантаження нових версій проекту та планування павуків. Ця функція є необов'язковою і її можна відключити. Scrapyd постачається з мінімальним веб-інтерфейсом (для моніторингу запущених процесів та доступу до журналів), доступ до якого можна отримати за адресою *http: // localhost: 6800 /*. Це значно спрощує розробку.

Для зв’язку також потрібно створити так званий *конвеєрний елемент* для скрапінгу. Конвеєр (“Pipepline”) - це клас для здійснення дій над спаршеними сутностями. З документації [14]:

Типовими напрямками використання конвеєрних елементів є:

- очищення даних HTML;
- перевірка скребкованих даних (перевірка наявності елементів у певних полях);
- перевірка наявності дублікатів (та їх видалення);
- зберігання скребленого елемента в базі даних;

Нас в першу чергу цікавить зберігання скребленого елемента в базі даних.

2.5 Підходи до розробки клієнтського інтерфейсу.

Для підвищення зручності використання програми важливо мати добре розроблений інтерфейс. «Вісім золотих правил дизайну інтерфейсу» Шнайдермана - це посібник щодо гарного дизайну інтерфейсів. Шнайдерман запропонував цей набір принципів, евристично виведених впродовж років досвіду та підходящих для

більшості інтерактивних систем після їх належного вдосконалення, розширення та інтерпретації. Ось їх перелік: [8]

- Прагніть до консистенції дизайну. Стандартизація способу передачі інформації забезпечує користувачам можливість будувати постійне уявлення про систему прямо в процесі ознайомлення з нею без необхідності вивчати нові представлення одних і тих же дій. Послідовність відіграє важливу роль, допомагаючи користувачам ознайомитися з цифровим представленням продукту для того, щоб вони могли легше досягати своїх цілей.

  “У подібних ситуаціях слід вимагати чітких послідовностей дій, у підказках, меню та довідкових екранах повинна використовуватися однакова термінологія з послідовними командами”.

- Частим користувачам слід дозволити використовувати ярлики. При збільшенні використання виникає потреба у більш швидких методах виконання завдань та зменшенні кількості взаємодій. Скорочення, функціональні клавіші, приховані команди та засоби макросу стануть у нагоді для завзятих користувачів.
- Робіть інформативні сповіщення. Користувачі постійно повинні знати, де вони перебувають і що відбувається. Для кожної дії повинні бути відповідні зручні для людини повідомлення протягом розумного проміжку часу. Для частих і незначних дій реакція може бути скромною, тоді як для рідкісних і серйозних дій відповідь повинна бути більш істотною.
- Діалогове вікно проектування для завершення роботи. Не змушуйте своїх користувачів гадати. Скажіть їм, до чого призвела їх дія.

“Інформаційний зворотний зв'язок після завершення групи дій дає користувачам задоволення від виконання мети, почуття полегшення, сигнал відкинути з розуму плани та варіанти надзвичайних ситуацій, і вказує на те, що шлях зрозумілий для підготовки до наступної групи дій”.

- Запропонуйте просту обробку помилок. Наскільки це можливо, спроектуйте систему, щоб користувач не міг зробити серйозну помилку. Якщо помилка допущена, система повинна бути в змозі виявити помилку і запропонувати прості, зрозумілі механізми поводження з помилкою.
- Дозвіл легкої відміни дій. Ця функція знімає тривогу, оскільки користувач знає, що помилки можна скасувати, це, в свою чергу заохочує вивчення незнайомих ситуацій, спонукає до ознайомлення з можливостями інтерфейсу системи. Одиницями оборотності можуть бути окремі дії, введення даних або цілі групи дій.
- Підтримка внутрішньої траєкторії контролю. Досвідчені юзери сильно бажають відчути, що вони відповідають за систему і що система реагує на їх дії. Створіть систему, щоб зробити користувачів ініціаторами дій, а не респондентами.
- Зменшення короткочасного навантаження на пам'ять. Обмеження обробки інформації людиною в короткостроковій пам'яті вимагає, щоб інтерфейси були максимально простими з належною ієрархією інформації та пріоретизували впізнання над згадуванням. Розпізнати щось завжди простіше, ніж згадати, тому що розпізнавання передбачає сприйняття так званих “візуальних підказок”, що допомагають нам віднайти у пам’яті за допогою асоціативного ряду необхідну інформацію.

Якоб Нільсен, відомий консультант із користування веб-сторінками та партнер групи Nielsen Norman Group, та Рольф Моліх, ще один відомий експерт з юзабіліті, встановили перелік десяти рекомендацій щодо дизайну користувальницького інтерфейсу у 1990-х роках. Необхідно зауважити, що між евристичними правилами Нільсена та Моліха і восьми золотими правилами Бена Шнайдермана існує значний збіг. Через те, що ці 10 правил в цілому скоріше розширюють вісім золотих правил Шнайдермана, нижче будуть представлені лише та частина інформації, що відмінна або являється розширенням концептів Шнайдермана, а саме: [7]

- Видимість стану системи. Користувачі завжди повинні бути поінформовані про операції в системі, з легким для розуміння та добре видимим статусом, що відображається на екрані протягом достатнього проміжку часу.
- Збіг між системою та реальним світом. Дизайнери повинні намагатися відображати мову та поняття, які користувачі знайдуть у реальному світі, виходячи з того, хто їх цільові користувачі. Представлення інформації в логічному порядку та вигляд інформації, що максимально виправдовує очікування користувачів, що випливають із їхнього досвіду в реальному світі, зменшить когнітивне навантаження та полегшить використання системи.
- Послідовність та стандарти. Дизайнери інтерфейсів повинні забезпечити підтримку як графічних елементів, так і термінології на подібних платформах. Наприклад, піктограма, що представляє одну категорію чи концепцію, не повинна представляти іншу концепцію, коли вона використовується на іншому екрані.

- Попередження помилок. По можливості проектуйте системи так, щоб потенційні помилки зводилися до мінімуму. Користувачі не люблять, щоб їх закликали виявляти та виправляти проблеми, які, можливо, виходять за рамки їх рівня знань. Усунення або позначення дій, які можуть призвести до помилок, є двома можливими засобами для запобігання помилкам.
- Зменшення кількості видимої інформації. Таким чином досягається зменшення когнітивного навантаження, так як необхідна інформація знаходиться у межах дисплея та не обтяжена зайвим, поки користувачі вивчають інтерфейс. Людська увага обмежена, і ми здатні одночасно підтримувати близько п'яти предметів у нашій короткотерміновій пам'яті. Тому занадто переповнений інтерфейс може стати серйозною перешкодою для користувача під час використання застосунку.
- Гнучкість та ефективність використання. При збільшенні використання виникає попит на меншу кількість взаємодій, що дозволяють швидше переходити. Цього можна досягти за допомогою скорочень, функціональних клавіш, прихованих команд та макро засобів. Користувачі повинні мати можливість налаштувати або адаптувати інтерфейс відповідно до своїх потреб, щоб часті дії можна було досягти більш зручними засобами.
- Естетичний та мінімалістичний дизайн. Зведення надлишковості до мінімуму. Так як вся непотрібна інформація конкурує з обмеженими ресурсами уваги, які можуть перешкоджати пошуку відповідної інформації в пам'яті. Отже, відображення повинно бути зведене лише до необхідних компонентів для поточних завдань, забезпечуючи при цьому

чітко видимі та однозначні засоби переходу до іншого контенту.

- Довідка та документація. В ідеалі ми хочемо, щоб користувачі орієнтувалися в системі, не звертаючись до документації. Однак, залежно від типу рішення, може знадобитися документація. Коли користувачі потребують допомоги, переконайтеся, що вона легко доступна, описує ситуацію, що склалася, та сформульована таким чином, щоб містити необхідні кроки для вирішення проблеми, з якою стикнулися користувачі.

Google Inc., багатомільярдна технологічна компанія, безумовно, проектує дизайни, що слідують вищевказаним правилам. Джон Вілей, головний дизайнер пошуку Google у 2012 році, сказав: "Коли я думаю про дизайн та створення чудового досвіду користувачів, я взагалі думаю про це з точки зору трьох речей: зручності, корисності та бажаності"[9]. Правила для розробників інтерфейсів, згадані вище, досить непогано висвітлюють ці три ключові компоненти користувальницького досвіду, а це означає, що непоганою ідеєю було б використати їх при розробці клієнтського інтерфейсу для застосунку.

2.6 Вибір технологічної бази для розробки користувацького інтерфейсу згідно правил та рекомендацій з пункту 2.5.

В контексті web-розробки з використанням шаблону MVC за інтерфейс клієнта відповідає View (Вид). Він ще іменується фронтендом (з англ. “front-end” — “передня частина”). Основними інструментами для розробки фронтенду є HTML, CSS та мова програмування JavaScript.

2.6.1 HTML

Hyper Text Markup Language (HTML) є основою будь-якого процесу розробки веб-сайтів, без якого веб-сторінка не існує. Мова розмітки вказує браузеру, що текст можна перетворити на зображення, таблиці, посилання та інші подання. Саме HTML-код забезпечує загальну правила того, як буде виглядати сайт. Остання версія HTML називається HTML5 і була опублікована 28 жовтня 2014 року. В контексті розробки front-end HTML зазвичай генерується шаблонно. В web-фреймворці Django, наприклад є ціла мова для спрощення переходу від данних на сервері до їх відображення у вікні браузера.

Django визначає стандартний API для завантаження та візуалізації шаблонів незалежно від бекенду. Завантаження складається з пошуку шаблону для даного ідентифікатора та попередньої його обробки, зазвичай його компіляції для представлення в пам'яті. Візуалізація означає інтерполяцію шаблону з контекстними даними та повернення отриманого рядка. Система шаблонів Django надає теги, які функціонують аналогічно деяким інструкціям програмування - тег *if* для булевих тестів, тег *for* для циклу тощо - але вони не просто виконуються як відповідний код Python, і система шаблонів не виконуватиме довільні вирази мови Python.

```
{% extends "base_generic.html" %}

{% block title %}{{ section.title }}{% endblock %}

{% block content %}
<h1>{{ section.title }}</h1>

{% for story in story_list %}
<h2>
  <a href="{{ story.get_absolute_url }}">
    {{ story.headline|upper }}
  </a>
</h2>
<p>{{ story.tease|truncatewords:"100" }}</p>
{% endfor %}
{% endblock %}
```

Рис. 2.3 — приклад файлу з шаблоном в Django. [10]

Шаблон - це текстовий файл. Він може генерувати будь-який текстовий формат (HTML, XML тощо). Шаблон містить змінні, які замінюються значеннями, коли шаблон оцінюється, та теги, які керують логікою шаблону. Також підтримується доступ до атрибутів об'єктів, фільтрація, блоки тощо. В цілому для статичних сторінок (як-от наприклад в контексті сайту web-агрегатора сторінок з окремими новинами, сторінки авторизації) функціоналу шаблонів Django має цілком вистачати.

2.6.2 CSS

Каскадні таблиці стилів (CSS) контролюють зовнішний вигляд - презентабельність сайту. Це робиться, за допомогою спеціальних таблиць стилів, що відповідають заздалегідь визначеним правилам стилів і відповідно змінюють вигляд елементів сайту в залежності від таких факторів, як розмір екрана пристрою, версії браузера, наявність чи відсутність певних шрифтів у системі користувача та ін.

```css
p {
   font-family: arial, helvetica, sans-serif;
}
h2 {
   font-size: 20pt;
   color: red;
   background: white;
}
.note {
   color: red;
   background-color: yellow;
   font-weight: bold;
}
p#paragraph1 {
   padding-left: 10px;
}
a:hover {
   text-decoration: none;
}
#news p {
   color: blue;
}
[type="button"] {
   background-color: green;
}
```

Рис. 2.4 — приклад CSS таблиці стилів.[11]

В контексті роботи з фреймворком Django CSS файли зазвичай розташовуються в окремій папці та довантажуються до HTML шаблонів за допомогою спеціальної команди *{% load static %}*.

2.6.3 JavaScript

JavaScript - імперативна мова програмування на основі подій (на відміну від декларативної мови HTML), яка використовується для перетворення статичної сторінки HTML в динамічний інтерфейс. JavaScript-код може використовувати Document Object Model (DOM), що надається стандартом HTML, для маніпулювання веб-сторінкою у відповідь на події, наприклад введення користувачем певної інформації.

Використовуючи техніку під назвою AJAX (англ. “Asynchronous JavaScript and XML” — “Асинхронний JavaScript та XML”), код JavaScript також може активно отримувати фоновий вміст з Інтернету (незалежно від оригінального отримання HTML-сторінок, без перезавантаження сторінки), а також реагувати на події зі сторони сервера, роблячи веб-сторінки по-справжньому динамічними.

Файли з JS кодом в ієрархії проекту Django зазвичай розміщуються в залежності від їх розміру або в шаблонах в тезі *<script></script>* або в окремих файлах в папці *static,* тоді принцип їх завантаження аналогічний довантаженню CSS файлів. Відповідно більшість веб-сайтів використовують сторонні бібліотеки JavaScript або веб-фреймворки для розробки скриптів на сторінках клієнтів. [12]

Серед бібліотек для розробки слід виділити в першу чергу *jQuery* - найпопулярнішу бібліотеку, яку використовують понад 70% веб-сайтів. На ній також базується більшість CSS та JavaScript фреймворків для веб-розробки, як-от *Bootstrap* та *React*. В цілому, використання як мінімум бібліотеки *jQuery* при розробці клієнтського інтерфейсу для веб-застосунку є дуже доречним, особливо якщо намагатись в повній мірі слідувати рекомендаціям з пункту 2.5.

ВИСНОВКИ ДО РОЗДІЛУ 2

В ході аналізу природи рішення проблеми побудови агрегатора новин на базі web-порталу стало зрозуміло, що найбільш підходящим по ряду причин способом реалізації всіх необхідних можливостей застосунку є шаблон проєктування MVC. При цьому за загальноприйнятою практикою в веб-розробці сайт матиме окремо серверну та клієнтську частину.

В якості фреймворку для розробки серверної частини був обраний python фреймворк Django. Модуль-парсер буде базуватись на python бібліотеці з відкритим кодом Scrapy, що зручно інтегрується в екосистему Django.

Щодо клієнтської частини, то слід зазначити, що важливо розробити інтерфейс користувача за “золотими” правилами дизайну. Тому простих статичних HTML-сторінок скоріш за все не вистачить, а значить доречно використати мову JavaScript (можливо використання відповідних бібліотек та фреймворків) для створення зручного та динамічного сайту.

3. ОПИС РОЗРОБКИ WEB-ПОРТАЛУ АГРЕГАЦІЇ НОВИН

В даному розділі помодульно представлена розробка web-порталу для агрегації новин з використанням python фреймворку Django. Основний принцип функціонування новинного агрегатора полягає в тому, що користувач задає деяку кількість посилань, з яких необхідно буде зчитувати оновлення. Після цього парсер автоматично переглядає задані посилання та наповнює базу данних статтями (що в RSS та Atom стрічках розміщуються в тегах *<item>*). Далі ці статті з бази можна за запитом клієнта відображати в зручному виді стрічок оновлень.

3.1 Загальна структура проєкту.

Як і будь-який веб сайт, портал для агрегації новин має складатися з клієнтської та серверної частини. На додачу до цього необхідний окремий сервер для модуля парсера. Наступна структурна схема системи являє собою більш деталізовану версію рис. 2.2.

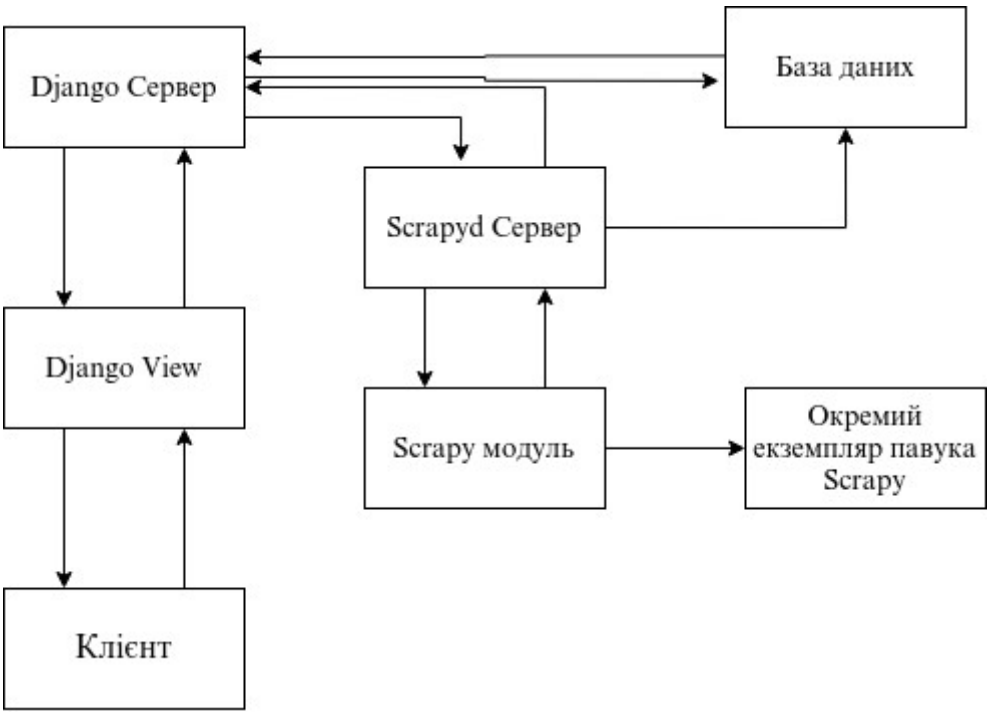


Рис. 3.1 — структурна схема системи.

Щодо алгоритму роботи системи, треба зазначити, що він практично повністю співвідноситься з теоретичним алгоритмом, поданим у другому пункті другого розділу. Щоправда, має місце один нюанс, а саме те, як має вести себе система в залежності від того, чи користувач має бажання отримати оновлення від заздалегідь обраних їм самим джерел або ж додати абсолютно нове джерело і одразу ж отримати всі статті звідти за певний період часу.

Для сценарію номер 1 (отримання “стрічки новин”) схема дії системи досить тривіальна:

1. клієнт (тут і далі під клієнтом мається на увазі авторизований користувач) заходить на сторінку зі своєю стрічкою (стрічка по своїй суті є колекцією джерел оновлень);
2. відновідний Django View робить запит з усіма джерелами, для яких необхідно отримати перелік статей на сервер;
3. сервер отримує статті для кожного з джерел, прицьому зазначає для кожного з них час оновлення;
4. сервер повертає до Django View колекцію об’єктів статей для кожного джерела;
5. Django View генерує шаблон для відображення поточної стрічки для клієнта.

Слід також зазначити, що актуальність оновлень джерел в базі може бути поганою при такому алгоритмі рішення задачі. Проте існують варіанти рішення данної проблеми, наприклад, встановити графік оновлень джерел з бази на сервері. В такому разі “свіжість” данних в базі буде в межах вікна графіку, яке можно зробити доволі малим, щоправда слід зазначити, що при збільшенні кількості джерел збільшується і навантаження як на сервер, так і на модуль-парсер, тому слід заздалегідь подбати про поступове (каскадне) оновлення інформації в базі. Іншим варіантом являється оновлення за запитом користувача, тобто для

кожного джерела клієнт матиме час останнього оновлення та опцію “Оновити”.

Повернемося до сценарію 2 (додавання нового джерела одразу ж оновленням бази по цьому джерелу), алгоритм якого наступний:

1. клієнт бажає додати нове джерело до своєї стрічки (вводить посилання на джерело до спеціального поля);
2. Django View зчитує це поле та перевіряє чи поле є валідною адресою URL та за умови корректності відправляє запит на сервер (у випадку некоректно заданої адреси View має повернути клієнту повідомлення про помилку);
3. сервер перевіряє чи є вже дане джерело в базі (можливо для іншого користувача вже воно парсилося), якщо так, то запускає процедуру оновлення (по-суті це означає що парсер замість фіксованого проміжку часу, за який треба знати статті, отримає дату та час найсвіжішої статті з бази та буде зберігати все, що свіжіше), якщо ж ні, то передає запит далі на Scrapyd сервер;
4. Scrapyd сервер запускає відповідно павука Scrapy (алгоритм роботи буде подано в наступних пунктах), результатом роботи якого стане потік об’єктів типу стаття до бази;
5. весь цей час сервер через визначені проміжки часу (5-10 с) перевіряє стан “роботи” звертаючись до Scrapyd сервера, а Django View теж через визначені проміжки часу аналогічним чином здійснює запит на Django сервер щодо статусу роботи;
6. цойно робота буде закінчена, сервер вивантажить з бази щойно оновлені статті та передаст їх у View;
7. View представить оновлену стрічку клієнту;

Як видно зі алгоритмів сценаріїв вище, основні операції системи досить прозорі та послідовні. Далі будуть викладені деталі реалізації кожного з елементів системи.

3.2 Модель данних.

Основною ланкою та виходить, що найменшою в представленні у базі є *NewsArticle* — новинна стаття. Проте існувати сама по собі стаття не може, тому вона зв'язана з *NewsSource* — джерелом новин. Джерела новин являються складовими *NewsFeed* — стрічки новин, що є в кожного *User* — користувача. Наглядно зв'язки продемонстровані на рис. 3.2.

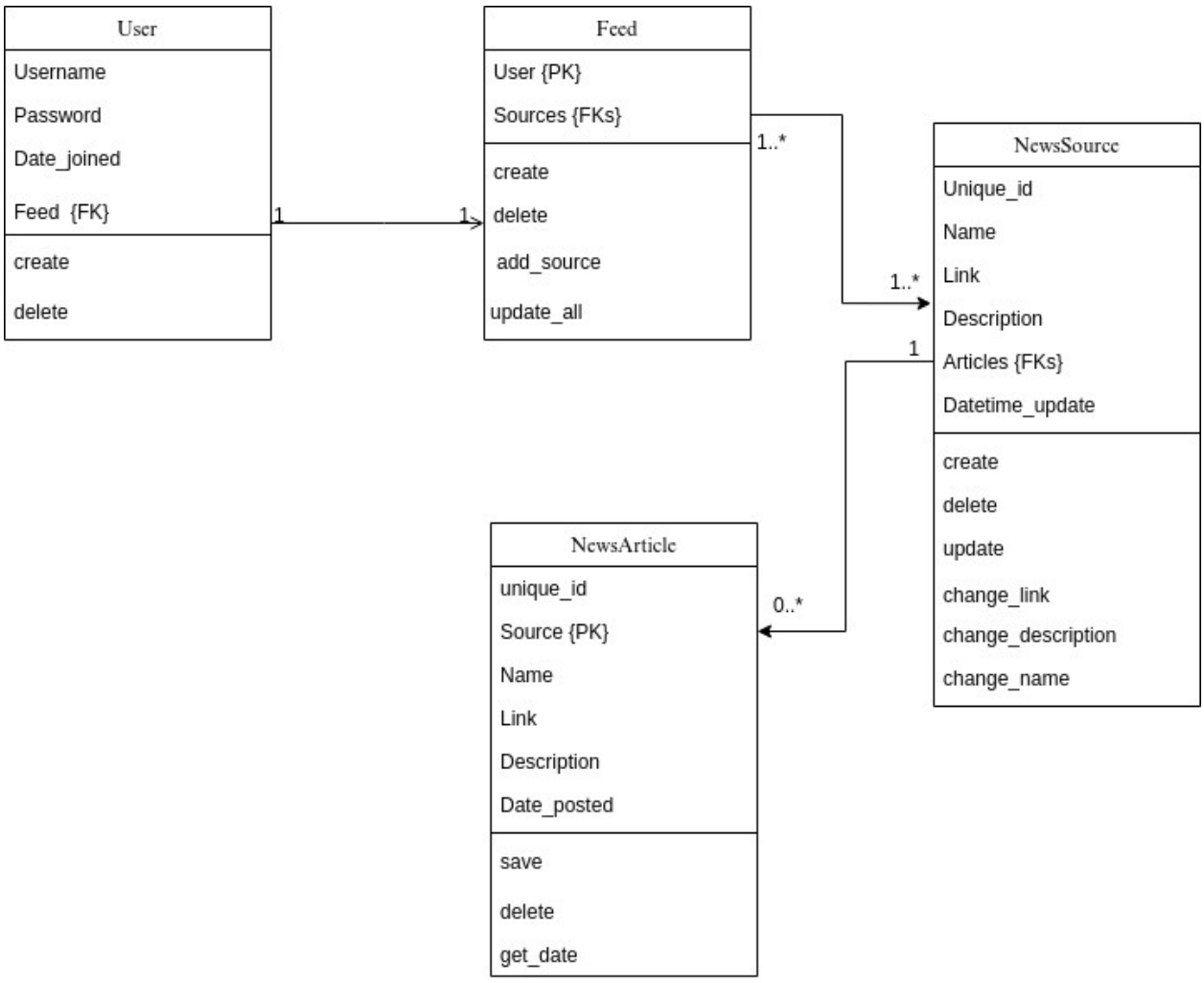


Рис. 3.2 — UML діаграма класів моделі django.

Приклад задавання моделі в вигляді класу для *NewsArticle* — новинної статті (методи опущені, так як геттери та сеттери для атрибутів, а також методи *save()* та *delete()* реалізовані у класі

django.db.models.Model за змовчуванням та наслідуються класом *NewsArticle*)

```
class NewsArticle(models.Model):
    # id (hash)?
    unique_id = models.CharField(max_length=100, null=True)
    article_name = models.CharField(max_length=200)
    article_link = models.URLField()
    article_description = models.TextField(max_length=400, blank=True)
    article_date_posted = models.DateTimeField("date published", default=timezone.now)
    article_source = models.CharField(max_length=100)
```

Рис. 3.3 — клас моделі *NewsArticle* — новинної статті.

3.3 Модуль-парсер.

Як вже було зазначено, модуль парсер має працювати зі стрічками в форматах RSS та Atom обов'язково, для виконання цієї умови непоганою ідеєю є наслідування павука Scrapy класу *XMLFeedSpider,* що має методи для парсингу XML-дерев. Крім того, досить значна частина сторінок з оновленнями складаються з гіперпосилань на стрічки з новинами певної категорії, тому при розробці павука необхідно враховувати цей момент. Відповідно алгоритм роботи модуля є наступним (рис. 3.4):

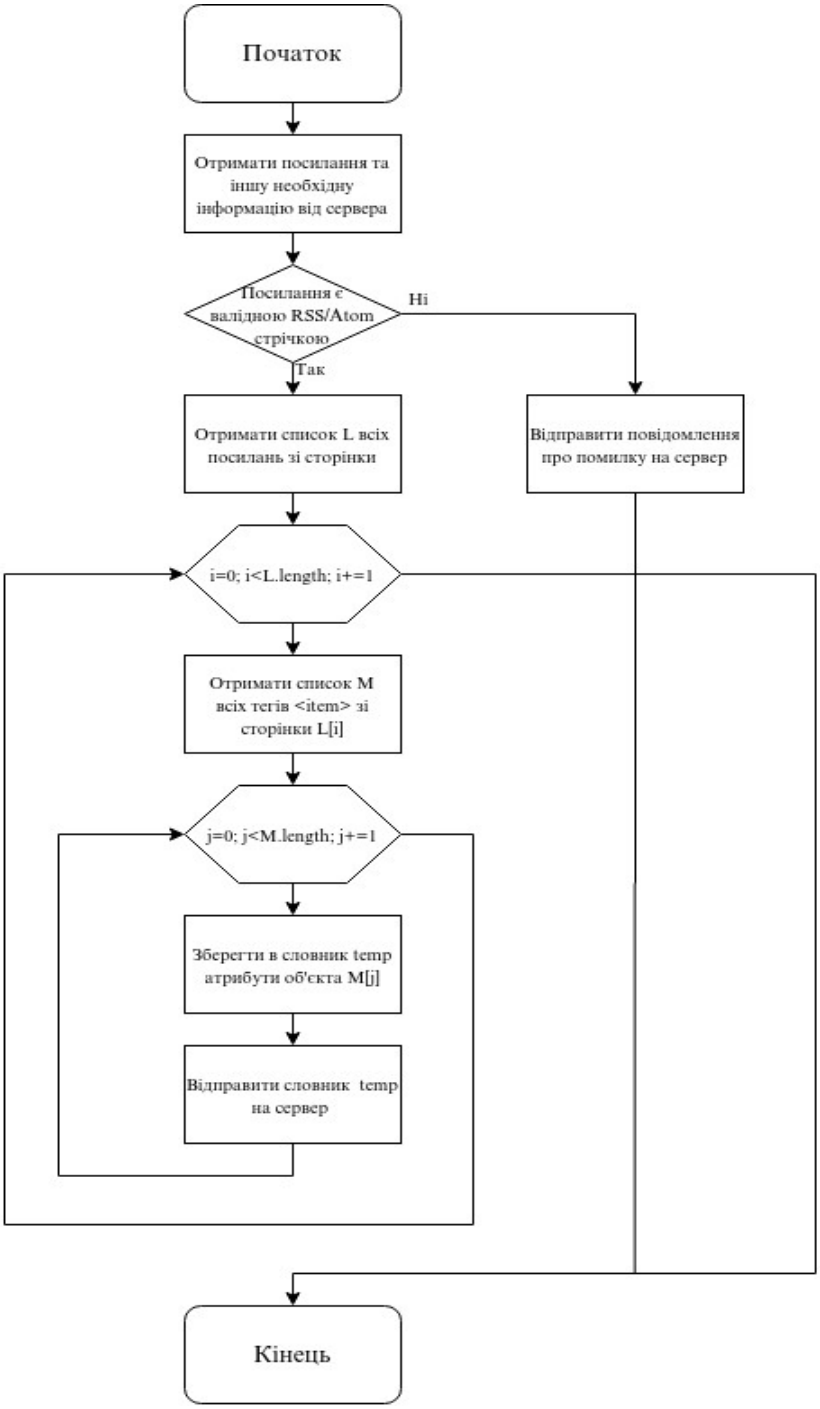


Рис. 3.4 — алгоритм роботи модуля-парсера.

В Scrapy є можливість реалізувати доволі зручним чином відправку “розібраних” елементів через сервер одразу до бази даних за допомогою так званого Pipeline — конвеєра.

```
i = {'url': item.xpath('link/text()').extract_first(),
     'title': item.xpath('title/text()').extract_first(),
     'description': item.xpath('description/text()').extract_first(),
     'date_publish': date_and_time,
     'site': self.allowed_domains[0]}
```

Рис. 3.5 — вигляд словника з елементом перед відправленням на сервер.

На рис. 3.5 ключами словника є рядки, що потім використовуються як значення для конструктора класу *NewsArticle* (викликається на рис. 3.6). Конструкція *item.xpath(...).extract_first()* повертає відповідний зміст з тегів (параметр методу *xpath*) у XML-дереві сторінки, що парситься.

```
def process_item(self, item, spider):

    article = NewsArticle(unique_id=self.unique_id, article_title=item.get('title'),
                          article_link=item.get('url'), article_description=item.get('description'),
                          article_publish_date=item.get('date_publish'),
                          article_site=item.get('site')
                          )
    if not NewsArticle.objects.filter(article_title=item.get('title')):
        article.save()
    return item
```

Рис. 3.6 — функція конвеєр для збереження елемента одразу до бази данних.

Функція на рис. 3.6 розпаковує словник з елементом, що віддається парсером, викликає конструктор класу *NewsArticle* та зберігає новостворений об’єкт до бази.

3.4 Django Views та проектування інтерфейсу користувача.

Керуючись порадами-правилами розробки дизайну для користувача, що були викладені в пункті 2.5 та беручи до уваги власний досить незначний досвід у проектуванні клієнтских інтерфейсів, була обрана наступна структура: мінімалістичний інтерфейс меню та навігаційної панелі (“нічого зайвого”), статичні сторінки реєстрації та

авторизації користувача, зроблені за допомогою html-шаблонів django та динамічна сторінка зі стрічкою користувача з функціоналом додавання, оновлення, видалення джерел та можливістю розширення опцій в майбутньому.

```
def login_request(request):
    if request.method == 'POST':
        form = AuthenticationForm(request=request, data=request.POST)
        if form.is_valid():
            username = form.cleaned_data.get('username')
            password = form.cleaned_data.get('password')
            user = authenticate(username=username, password=password)
            if user is not None:
                login(request, user)
                messages.info(request, f"You are now logged in as {username}.")
                return redirect("MainPage:homepage")
            else:
                messages.error(request, "Invalid username or password.")
        else:
            messages.error(request, "Invalid username or password.")
    form = AuthenticationForm()
    return render(request,
                  "MainPage/login.html",
                  {"form": form})
```

Рис. 3.7 — функція View, що відповідає за авторизацію користувача.

Як видно з рисунку 3.7, функція опрацьовує запит на авторизацію від користувача, отримує введені користувачем дані та перевіряє їх правильність. При цьому як у випадку помилки, так і при вдалій аутентифікації користувач побачить відповідне повідомлення. Функція вертає метод *render* — методом, що повертає HttpResponce — відповідь, що була побудована згідно шаблону, переданного параметром (на рис 3.7 це *“login.html”*).

```
{% extends 'MainPage/header.html' %}

{% block content %}

    <div class="container">
        <form method="POST">
        {% csrf_token %}
        {{form.as_p}}
            <button style="background-color:blue; color:yellow" class="btn btn-outline-info" type="submit">Login</button>
        </form>
        Don't have an account? <a href="/register" target="blank"><strong>register here</strong></a>!
    </div>
{% endblock %}
```

Рис. 3.8 — шаблон сторінки login.html.

На рисунку 3.8 можна побачити одну зі сторінок, а саме сторінку з формою для авторизації користувача, *{% extends 'MainPage/header.html' %}* означає, що частина коду розмітки сторінки береться з іншого файлу, в цьому випадку в файлі *header.html* знаходиться опис меню панелі навігації, а також імпорти необхідних бібліотек та файлів.

ВИСНОВКИ ДО РОЗДІЛУ 3

В данному розділі були викладені деталі розробки web-порталу агрегатора новин з вибірковими прикладами. Була представлена пара типових сценаріїв використання системи.

Щодо процесу розробки була показана структурна схема системи, блок-схема алгоритму роботи модуля парсера. В якості прикладу продемонстрована робота scrapy конвеєра для вивантаження отриманих новин до бази данних. Для демонстрації принципу функціонування клієнтського інтерфейсу в якості прикладу продемонстрована функція для рендерингу html-шаблону сторінки авторизації користувача.

4. ТЕСТУВАННЯ РОЗРОБКИ ТА ПРИКЛАД ЇЇ ВИКОРИСТАННЯ.

В цьому розділі буде покроково розглянуто з прикладами функціонал розробленого веб-застосунку.

Для використання порталу обов'язково потрібна реєстрація, тому спочатку продемонструємо вигляд сторінки для реєстрації та вигляд сторінки для авторизації.

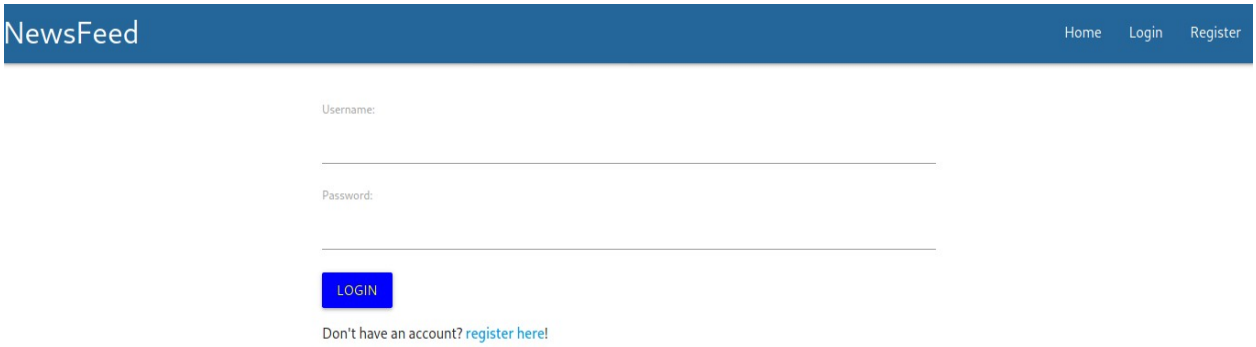


Рис. 4.1 — вид сторінки авторизації.

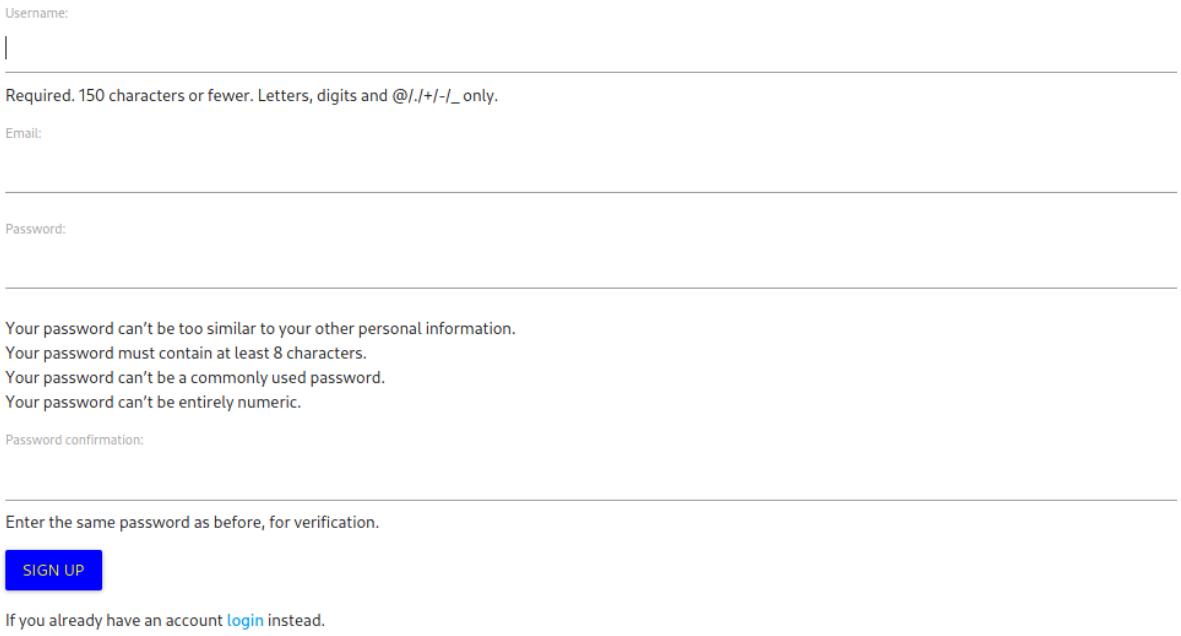


Рис. 4.2 — вид сторінки авторизації.

Після реєстрації система автоматично авторизує користувача та перенаправляє на домашню сторінку (для редагування власної стрічки новин).

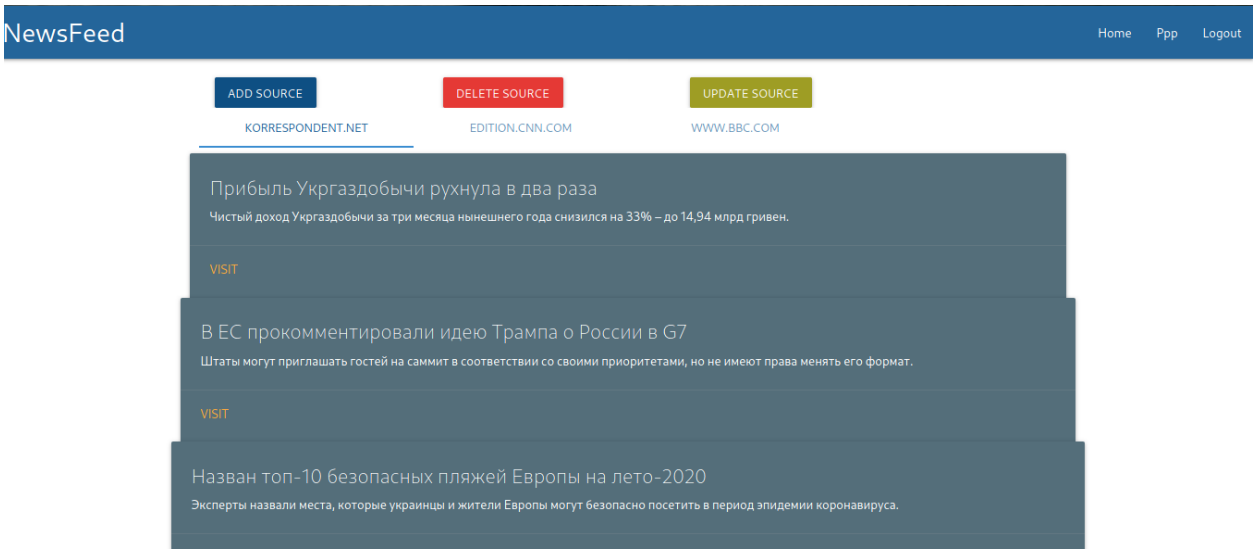


Рис. 4.3 — приклад вигляду стрічки новин.

На рисунку 4.3 можна побачити типовий вигляд стрічки новин. Згори є три кнопки, що реалізують наступний функціонал — зліва-направо: “Додати джерело”, “Видалити джерело” та “Оновити джерело”.

Щодо того, що відбувається з модулем-парсером під час роботи, то це можна дізнатись за допомогою зручних журналів модуля Scrapyd в веб-консолі (рис. 4.4 та 4.5).

Jobs

Go back

Project	Spider	Job	PID	Start	Runtime	Finish	Log
Pending							
Running							
default	korrespondent	0576cef8aa1c11eaa1cd240a648952df	9492	2020-06-09 09:39:53	0:00:10		Log
Finished							

Рис. 4.4 — вигляд веб-консолі scrapyd зі статусом для кожної “роботи”.

```
2020-06-02 11:24:28 [scrapy.core.scraper] DEBUG: Scraped from <200 http://k.img.com.ua/rss/ru/mainbyday.xml>
{'url': 'https://korrespondent.net/ukraine/4235837-v-krymu-za-hod-v-dva-raza-obmelely-vodokhranylyscha', 'title': 'Đ’ ĐšÑ€Ñ‹Đ¼Ñƒ Đ·Đ° Đ²Đ¾Đ´ Đ²Đ°Đ° Ñ€Đ°Đ·Đ° Đ¾Đ±Đ¼ĐµĐ»Đ¸Đ¾, Đ²Đ¾Đ´Đ¾Ñ…Ñ€Đ¾Đ¸Đ»Đ¸Ñ‰Đ°', 'description': '<img align="left" vspace="5" hspace="10" src="https://kor.ill.in.ua/m/190x120/2508578.jpg">Đ˜Ñ‚Đ¾Đ³Đ¸ Đ·Đ°Đ¿Đ°Ñ‚Đ¾Đ² Đ²Đ¾Đ´Đ¾Ñ‚ Đ¿Đ¾Đ¿Đ¾Đ»Đ¾Ñ‚Đ¾Đ¼, Đ²Đ¾Đ´Đ°Đ»Đ°ÑÑŒ Đ¾Đ´Đ½Đ¾Đ¹ Đ²Đ°Đ¶Đ½Đ¾Đ¹ Đ¿Đ¾Đ±Đ¾Ñ‚Đ¾Đ¹ Đ¾ĐºĐ¾Đ»Đ¾ 80 % Đ½Đ¾Đ²ĐµĐ½Đ¸Đ¼ Đ½Đ¾Ñ€Đ¼Ñ‹.', 'date_publish': datetime.datetime(2020, 6, 2, 8, 41, tzinfo=datetime.timezone(datetime.timedelta(seconds=10800))), 'site': 'korrespondent.net'}
2020-06-02 11:24:30 [scrapy.core.engine] DEBUG: Crawled (200) <GET http://k.img.com.ua/rss/ru/eurovision.xml> (referer: None)
2020-06-02 11:24:30 [scrapy.core.engine] INFO: Closing spider (finished)
2020-06-02 11:24:30 [scrapy.statscollectors] INFO: Dumping Scrapy stats:
{'downloader/request_bytes': 19892,
 'downloader/request_count': 71,
 'downloader/request_method_count/GET': 71,
 'downloader/response_bytes': 13689990,
 'downloader/response_count': 71,
 'downloader/response_status_count/200': 71,
 'elapsed_time_seconds': 6.345925,
 'finish_reason': 'finished',
 'finish_time': datetime.datetime(2020, 6, 2, 11, 24, 30, 551360),
 'item_scraped_count': 407,
 'log_count/DEBUG': 479,
 'log_count/INFO': 10,
 'memusage/max': 69058560,
 'memusage/startup': 69058560,
 'request_depth_max': 1,
 'response_received_count': 71,
 'scheduler/dequeued': 71,
 'scheduler/dequeued/memory': 71,
 'scheduler/enqueued': 71,
 'scheduler/enqueued/memory': 71,
 'start_time': datetime.datetime(2020, 6, 2, 11, 24, 24, 205435)}
2020-06-02 11:24:30 [scrapy.core.engine] INFO: Spider closed (finished)
```

Рис. 4.5 — приклад типового вмісту файлу-журналу завдання.

На рисунку 4.5 представлений завершення роботи модуля, зі статистикою щодо виконаного завдання. Функція ведення журналу може стати дуже у нагоді для програміста при пошуку несправностей та для аналізу навантаження на модуль.

ВИСНОВКИ ДО РОЗДІЛУ 4

В данному розділі було швидко оглянуто результат розробки агрегатора новин, реалізованого у вигляді web-порталу.

Були продемонстровані сторінки реєстрації та авторизації користувача — одна з необхідних умов персонального агрегатора новин, мінімалістичний вигляд стрічки з декількома джерелами новин.

Також була зазначена (з прикладом) зручна властивість scrapyd сервера, а саме можливість переглядати журнал для кожного завдання, отриманого модулем-парсером. Це дуже спрощує відладку системи.

ВИСНОВКИ

В представленому дипломному проекті була зроблена спроба вирішити проблему агрегації новин (в загальному випадку будь-яких оновлень сайту/каналу). Були розлянуті основні формати стрічок оновлень на сайтах, а саме RSS та Atom. Оскільки формою кінцевої реалізації обраний веб-сайт агрегатор, то відповідно розглянуто приклади сайтів-агрегаторів з виділенням основних властивостей кожного з них.

Далі було преставлено інструментарій для розробки з обгрунтуванням причин вибору певних знарядь. Для бази реалізації серверу був обраний python фреймворк Django, бібліотека Scrapy була обрана як основа для проектування модуля-парсера, клієнтський інтерфейс був розроблений в мінімалістичному стилі згідно рекомендацій по дизайну з використанням шаблонів django та JavaScript скриптів для стрічки оновлень.

Основні функції агрегатору новин, такі як можливість зробити персональну стрічку, актуальність новин, підтримку як RSS, так і Atom стрічок були реалізовані. Надалі доцільно буде сконцентруватися на так званих QoL (англ. “Quality-of-Life” — буквально “якість життя”) властивостях, тобто покращенні досвіду користувача при використанні системи. Наприклад покращеній фільтрації, додаванню варіантів вибору відображення оновлень, додаванню соціальних можливостей тощо.

ПЕРЕЛІК ПОСИЛАНЬ

1. Lash A. W3C takes first step toward RDF spec [Електронний ресурс] / Alex Lash. – 1997. – Режим доступу до ресурсу: https://web.archive.org/web/20110809151456/http://news.cnet.com/2100-1001-203893.html.
2. Trott B. Why We Need Echo [Електронний ресурс] / Benjamin Trott. – 2003. – Режим доступу до ресурсу: https://web.archive.org/web/20080216234454/http://www.sixapart.com/about/news/2003/06/why_we_need_ech.html.
3. Festa P. Dispute exposes bitter power struggle behind Web logs [Електронний ресурс] / Paul Festa. – 2003. – Режим доступу до ресурсу: https://web.archive.org/web/20090806234534/http://news.cnet.com/Battle-of-the-blog/2009-1032_3-5059006.html.
4. Atom Wiki. RSS and Atom compared [Електронний ресурс] / Atom Wiki – Режим доступу до ресурсу: http://www.intertwingly.net/wiki/pie/Rss20AndAtom10Compared.
5. Офіційна документація бібліотеки Scrapy [Електронний ресурс] – Режим доступу до ресурсу: https://docs.scrapy.org/.
6. Офіційна документація фреймворку Django версії 3.0 [Електронний ресурс] – Режим доступу до ресурсу: https://docs.djangoproject.com/en/3.0/.
7. Nielsen J. 10 Usability Heuristics for User Interface Design [Електронний ресурс] / J. Nielsen, R. Molich. – 1994. – Режим доступу до ресурсу: https://www.nngroup.com/articles/ten-usability-heuristics/.
8. Designing the User Interface: Strategies for Effective Human-Computer Interaction (5th Edition) / B.Shneiderman, C. Plaisant, M. Cohen, S. Jacobs., 2009. – 624 с.

9. Ryan R. The 3 Elements of Good Design [Електронний ресурс] / Riddle Ryan. – 2012. – Режим доступу до ресурсу: https://zurb.com/blog/the-3-elements-of-good-design-usability-u.
10. Django Template language [Електронний ресурс] – Режим доступу до ресурсу: https://docs.djangoproject.com/en/3.0/ref/templates/language/.
11. Огляд можливостей CSS [Електронний ресурс] – Режим доступу до ресурсу: https://www.tutorialspoint.com/css/index.htm.
12. Usage statistics of JavaScript libraries for websites [Електронний ресурс] – Режим доступу до ресурсу: https://w3techs.com/technologies/overview/javascript_library.
13. Hajba G. L. Website Scraping with Python: Using BeautifulSoup and Scrapy / Gábor László Hajba., 2015. – 241 c. – (1st).
14. Scrapy Item Pipeline [Електронний ресурс] – Режим доступу до ресурсу: https://docs.scrapy.org/en/latest/topics/item-pipeline.html.
15. Wilson J. L. Feedly Review [Електронний ресурс] / Jeffrey L. Wilson – Режим доступу до ресурсу: https://www.pcmag.com/reviews/feedly.
16. Tedeshi B. There's a Popular New Code for Deals: RSS [Електронний ресурс] / Bob Tedeshi. – 2006. – Режим доступу до ресурсу: https://web.archive.org/web/20060717202415/http://travel2.nytimes.com/2006/01/29/travel/29prac.html.
17. Qun W. Normalization and differentiation in Google News: a multi-method analysis of the world's largest news aggregator : дис. докт. / Qun Wang, 2020. – 256 c.
18. Perez S. Google News gets an AI-powered redesign [Електронний ресурс] / Sarah Perez. – 2018. – Режим доступу до ресурсу: https://techcrunch.com/2018/05/08/google-news-gets-an-ai-powered-redesign/.
19. Old Reader product tour [Електронний ресурс] – Режим доступу до ресурсу: https://theoldreader.com/pages/tour.

20. Greenfeld D. R. Two Scoops of Django: Best Practices for Django 1.8 / D. R. Greenfeld, A. R. Greenfeld., 2015. – 532 с.

Додатки

ДОДАТОК 1 – СИРЦЕВИЙ КОД ПРОЄКТУ

```
from django import forms
from django.contrib.auth.forms import UserCreationForm
from django.contrib.auth.models import User


class NewUserForm(UserCreationForm):
    email = forms.EmailField(required=True)

    class Meta:
        model = User
        fields = ("username", "email", "password1", "password2")

    def save(self, commit=True):
        user = super(NewUserForm, self).save(commit=False)
        user.email = self.cleaned_data["email"]
        if commit:
            user.save()
        return user


class FeedForm(forms.Form):
    url = forms.URLField(label='site:', max_length=100)


from django.db import models
from django.utils import timezone
import json
# Create your models here.


class NewsArticle(models.Model):
    # id (hash)?
    unique_id = models.CharField(max_length=100, null=True)
    article_title = models.CharField(max_length=200)
```

```
    article_link = models.URLField()
    article_description = models.TextField(max_length=400, blank=True)
           article_publish_date   =   models.DateTimeField("date   published",
default=timezone.now)
    article_site = models.CharField(max_length=100)

    @property
    def to_dict(self):
        data = {
            'title': self.article_title,
            'date': self.article_publish_date
        }
        return data

    def __str__(self):
        return self.article_title



from django.shortcuts import render, redirect
from django.http import HttpResponse
from .models import NewsArticle
from django.contrib.auth.forms import AuthenticationForm
from .forms import NewUserForm
from django.contrib.auth import logout, authenticate, login
from django.contrib import messages
from .forms import FeedForm
##
from uuid import uuid4
from urllib.parse import urlparse
from django.core.validators import URLValidator
from django.core.exceptions import ValidationError
from django.views.decorators.http import require_POST, require_http_methods
from django.shortcuts import render
from django.http import JsonResponse
from django.views.decorators.csrf import csrf_exempt
from scrapyd_api import ScrapydAPI
```

```
#from .utils import URLUtil
from .models import ScrapyItem

# connect scrapyd service
scrapyd = ScrapydAPI('http://localhost:6800')



def is_valid_url(url):
    validate = URLValidator()
    try:
        validate(url)  # check if url format is valid
    except ValidationError:
        return False
    return True



@csrf_exempt
@require_http_methods(['POST', 'GET'])  # only get and post
def crawl(request):
    # Post requests are for new crawling tasks
    if request.method == 'POST':

        url = request.POST.get('url', None)  # take url comes from client.

        if not url:
            return JsonResponse({'error': 'Missing  args'})

        if not is_valid_url(url):
            return JsonResponse({'error': 'URL is invalid'})

           domain = urlparse(url).netloc   # parse the url and extract the
domain
        unique_id = str(uuid4())  # create a unique ID.
```

```
        settings = {
            'unique_id': unique_id,  # unique ID for each record for DB
                    'USER_AGENT': 'Mozilla/5.0 (compatible; Googlebot/2.1;
+http://www.google.com/bot.html)'
        }
        if NewsArticle.objects.filter(article_site=domain):
                                                            date_latest     =
NewsArticle.objects.filter(article_site=domain).order_by('-
article_publish_date')[0].to_dict['date']
        else:
            date_latest = None

        task = scrapyd.schedule('default', 'korrespondent',
                                    settings=settings, url=url, domain=domain,
date_latest=date_latest)
             return JsonResponse({'task_id': task, 'unique_id': unique_id,
'status': 'started'})

    # Get requests are for getting result of a specific crawling task
    elif request.method == 'GET':
        task_id = request.GET.get('task_id', None)
        unique_id = request.GET.get('unique_id', None)

        if not task_id or not unique_id:
            return JsonResponse({'error': 'Missing args'})

        status = scrapyd.job_status('default', task_id)
        if status == 'finished':
            try:
                item = ScrapyItem.objects.get(unique_id=unique_id)
                return JsonResponse({'data': item.to_dict['data']})
            except Exception as e:
                return JsonResponse({'error': str(e)})
        else:
            return JsonResponse({'status': status})
```

```
def homepage(request):
    return render(request=request,
                  template_name='MainPage/home.html',
                  context={'news_articles': NewsArticle.objects.all})

def register(request):
    if request.method == 'POST':
        form = NewUserForm(request.POST)
        if form.is_valid():
            user = form.save()
            username = form.cleaned_data.get('username')
            messages.success(request, f"Hooray, {username} is created!")
            login(request, user)
            messages.info(request, f"You're now logged in as {username}.")
            return redirect("MainPage:homepage")
        else:
            for msg in form.error_messages:
                                            messages.error(request,   f'{msg}:
{form.error_messages[msg]}')

            return render(request,
                          'MainPage/register.html',
                          {'form': form})

    form = NewUserForm
    return render(request=request,
                  template_name="MainPage/register.html",
                  context={"form": form})


def logout_request(request):
    logout(request)
    messages.info(request, "Logged out.")
    return redirect("MainPage:homepage")
```

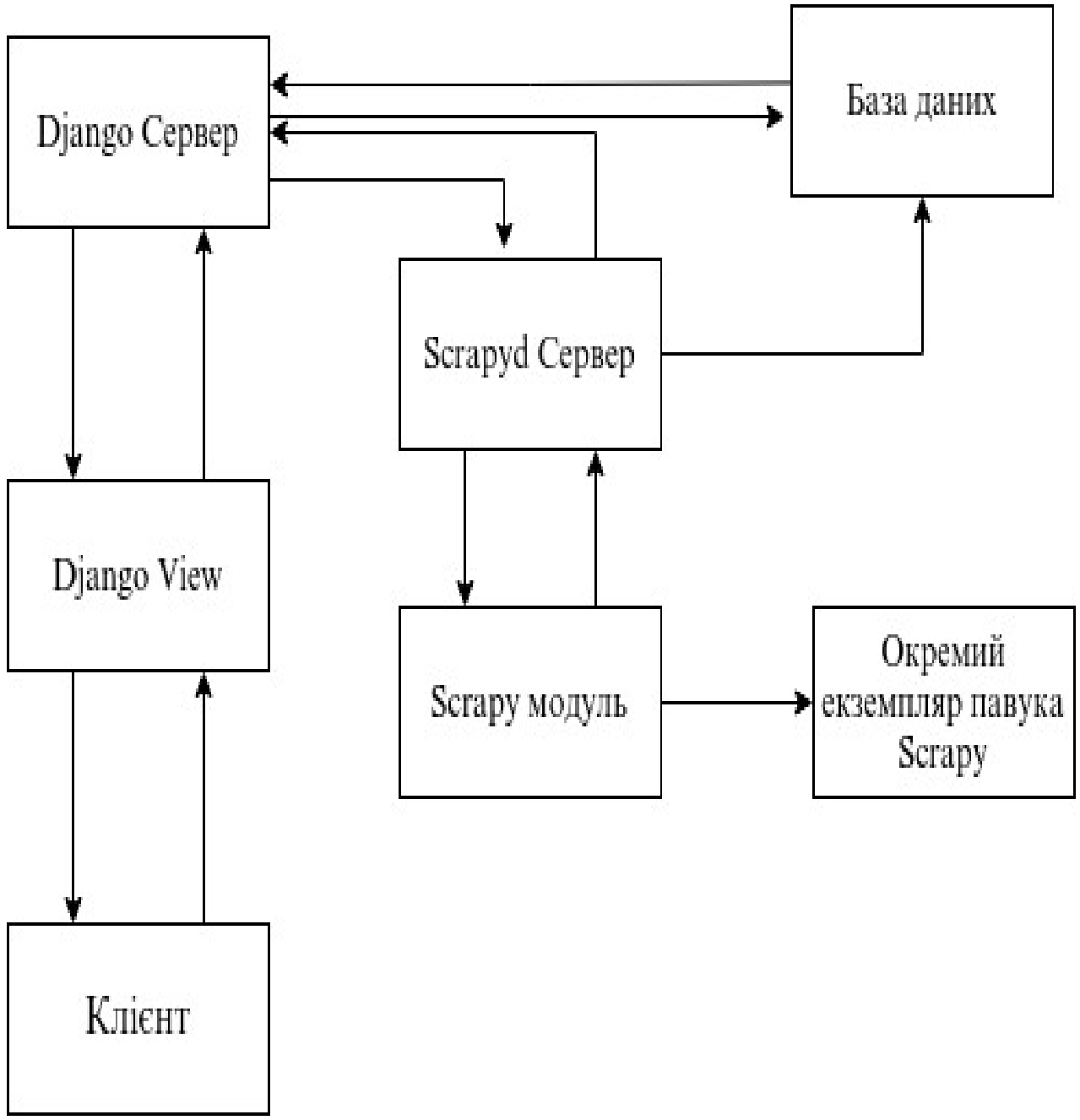
```
def login_request(request):
    if request.method == 'POST':
        form = AuthenticationForm(request=request, data=request.POST)
        if form.is_valid():
            username = form.cleaned_data.get('username')
            password = form.cleaned_data.get('password')
            user = authenticate(username=username, password=password)
            if user is not None:
                login(request, user)
                        messages.info(request, f"You are now logged in as
{username}.")
                return redirect("MainPage:homepage")
            else:
                messages.error(request, "Invalid username or password.")
        else:
            messages.error(request, "Invalid username or password.")
    form = AuthenticationForm()
    return render(request,
                  "MainPage/login.html",
                  {"form": form})


from MainPage.models import ScrapyItem, NewsArticle
import json
from django.utils import timezone
import datetime


class ScrapyAppPipeline(object):
    def __init__(self, unique_id, *args, **kwargs):
        self.unique_id = unique_id
        self.items = []

    @classmethod
    def from_crawler(cls, crawler):
```

```
        return cls(
               unique_id=crawler.settings.get('unique_id'),  # this will be
passed from django view
        )

    def process_item(self, item, spider):

                          article   =   NewsArticle(unique_id=self.unique_id,
article_title=item.get('title'),
                                                 article_link=item.get('url'),
article_description=item.get('description'),
                               #article_publish_date=datetime.datetime.strpti
me(item.get('date_publish'),
                                                                          #
"%a, %d %b %Y %H:%M:%S %Z"),
                               article_publish_date=item.get('date_publish'),
                               article_site=item.get('site')
                               )
        if not NewsArticle.objects.filter(article_title=item.get('title')):
                                  #   quote   =   Quote(text=item.get('text'),
author=item.get('author'))
            article.save()
        return item


from scrapy.spiders import XMLFeedSpider
import scrapy, datetime
import pytz
import logging


class KorrespondentSpider(XMLFeedSpider):
    """
    Cascade Spider for crawling rss feeds with categories
    using korrespondent.net rss feed as an example
    """
```

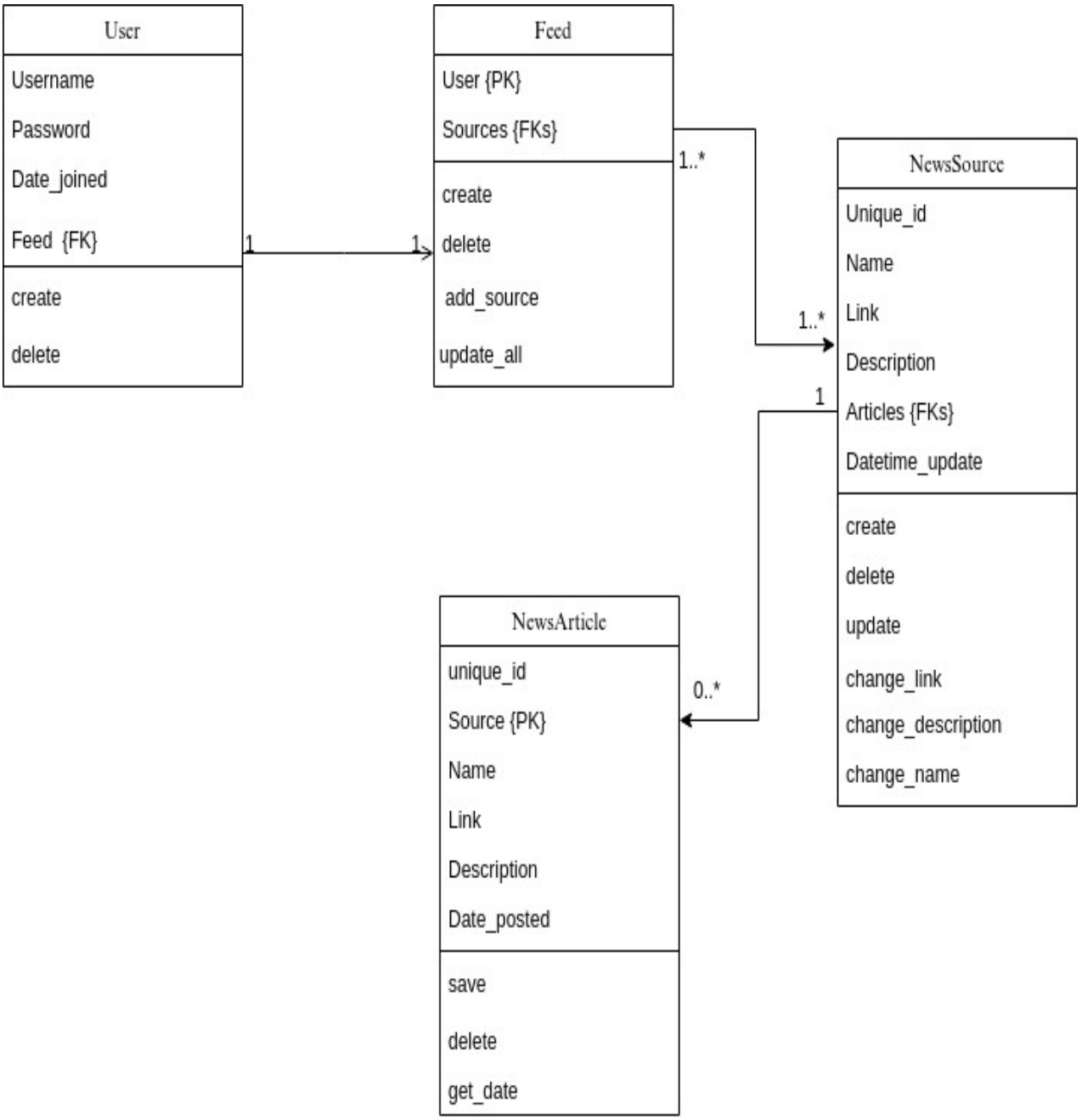
```
    name = 'korrespondent'
    iterator = 'iternodes'
    itertag = 'item'

    def __init__(self, *args, **kwargs):
        self.url = kwargs.get('url')
        self.domain = kwargs.get('domain')
        self.start_urls = [self.url]
        self.allowed_domains = [self.domain]
        if kwargs.get('date_latest'):
            self.l_date = kwargs.get('date_latest')
             self.l_date = datetime.datetime.strptime(self.l_date, "%Y-%m-%d
%H:%M:%S%z")
        else:
               self.l_date = datetime.datetime.now(datetime.timezone.utc) -
datetime.timedelta(days=7)

        super(KorrespondentSpider, self).__init__(*args, **kwargs)


    def parse(self, response):
        """
        Extracts the Rss Feed and initiates crawling it.
        Crawls category tree
        :param obj response: XMLResponse
        """
        def try_parsing_date(text):
             for fmt in ("%a, %d %b %Y %H:%M:%S %Z", "%a, %d %b %Y %H:%M:%S
%z"): #"%a, %d %b %Y %H:%M:%S %z",
                try:
                    return datetime.datetime.strptime(text, fmt)
                except ValueError:
                    pass
            raise ValueError('no valid date format found')
        for item in response.xpath('//item'):
            if item.xpath('pubDate/text()').extract_first():
```

```
                                                        date_and_time    =
try_parsing_date(item.xpath('pubDate/text()').extract_first())
                if not date_and_time.tzinfo:
                    date_and_time = pytz.UTC.localize(date_and_time)
                if date_and_time > self.l_date:
                    i = {'url': item.xpath('link/text()').extract_first(),
                        'title': item.xpath('title/text()').extract_first(),
                                                            'description':
item.xpath('description/text()').extract_first(),
                         'date_publish': date_and_time, #item.xpath('pubDate/
text()').extract_first(),
                         'site': self.allowed_domains[0]}
                    yield I

        lists_of_rss_feeds = response.css('a').extract()
        for i in range(len(lists_of_rss_feeds)):
            lists_of_rss_feeds[i] = lists_of_rss_feeds[i].split('"')[1]
            if lists_of_rss_feeds[i][-4:] == '.xml':
                                yield  scrapy.Request(lists_of_rss_feeds[i],
dont_filter=True, callback=self.parse_mode)
            elif lists_of_rss_feeds[i][-4:] == '.rss':
                                yield  scrapy.Request(lists_of_rss_feeds[i],
dont_filter=True, callback=self.parse_mode)

    def parse_mode(self, response):
        """
        Second wave of crawling - crawling categories
        :param response: XMLResponse
        :return: item
        """
        def try_parsing_date(text):
             for fmt in ("%a, %d %b %Y %H:%M:%S %Z", "%a, %d %b %Y %H:%M:%S
%z"): #"%a, %d %b %Y %H:%M:%S %z",
                try:
                    return datetime.datetime.strptime(text, fmt)
                except ValueError:
```

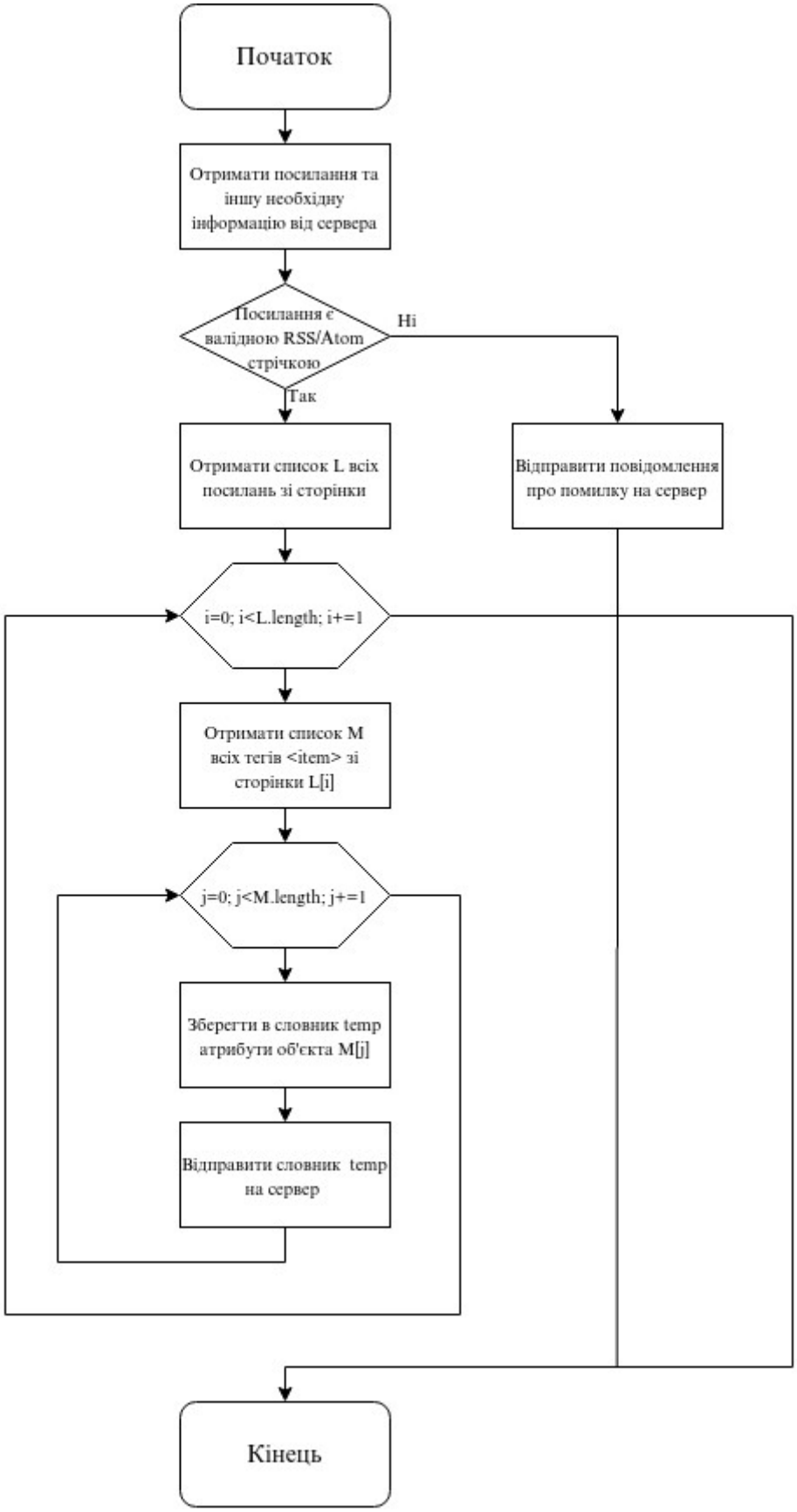
```
                    pass
            raise ValueError('no valid date format found')
        for item in response.xpath('//item'):
                                                    date_and_time     =
try_parsing_date(item.xpath('pubDate/text()').extract_first())
            if not date_and_time.tzinfo:
                date_and_time = pytz.UTC.localize(date_and_time)
            if date_and_time > self.l_date:
                i = {'url': item.xpath('link/text()').extract_first(),
                     'title': item.xpath('title/text()').extract_first(),
                                                          'description':
item.xpath('description/text()').extract_first(),
                     'date_publish': date_and_time,
                     'site': self.allowed_domains[0]}
                yield i
```



					ІА/ЛЦ.467200.006 Д2		
Змн.	Арк.	№ докум.	Підп.	Дата			
Розробив		Надолінський			Агрегатор новин, реалізований у вигляді Web-порталу Структурна схема системи	Літ.	Аркуш 1 / Аркушів 1
Перевірив		Антонюк					
Норм. контр.		Сімоненко				НТУУ «КПІ» ФІОТ ІО-63	
Затв.		Стіренко					



					ІА/ЛЦ.467200.007 ДЗ		
Змн.	Арк.	№ докум.	Підп.	Дата			
Розробив		Надолінський			Агрегатор новин, реалізований у вигляді Web-порталу UML діаграма класів моделі бази данних	Літ.	Аркуш 1 / Аркушів 1
Перевірив		Антонюк					
Норм. контр.		Сімоненко				НТУУ «КПІ» ФІОТ ІО-63	
Затв.		Стіренко					



					ІА/ЛЦ.467200.008 Д4	
Змн.	Арк.	№ докум.	Підп.	Дата		
Розробив		Надолінський			Агрегатор новин, реалізований у вигляді Web-порталу Блок-схема алгоритму роботи модуля-парсера	Літ. / Аркуш 1 / Аркушів 1
Перевірив		Антонюк				
Норм. контр.		Сімоненко				НТУУ «КПІ» ФІОТ ІО-63
Затв.		Стіренко				